

CZECH TECHNICAL UNIVERSITY  
FACULTY OF ELECTRICAL ENGINEERING  
DEPARTMENT OF COMPUTER SCIENCE



# **A Modern Wiki Web application based on User Centered Design method**

BACHELOR'S THESIS

**Marek Dluhoš**

Prague, May 2019



## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Dlugoš** Jméno: **Marek** Osobní číslo: **453023**  
Fakulta/ústav: **Fakulta elektrotechnická**  
Zadávající katedra/ústav: **Katedra počítačů**  
Studijní program: **Softwarové inženýrství a technologie**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Moderní wiki web aplikace za využití metody User Centered Design**

Název bakalářské práce anglicky:

**A Modern Wiki Web application based on User Centered Design method**

Pokyny pro vypracování:

Wiki software je v dnešní době hojně využíván pro sdílení informací napříč organizacemi a je téměř nevyhnutnou součástí každé větší organizace, která potřebuje poskytnout určité informace určité skupině lidí.

Na trhu s wiki softwarem, ale dosud není vidět dostatečná odezva aktuálních trendů v oblasti webových aplikací. Dokonce ani v oblasti open source softwarů, kde se již k ostatním zastaralým řešením objevily nové alternativy.

Cíle této práce jsou:

1. Provést uživatelský výzkum a kompetitivní analýzu. Identifikovat aktuální problémy náhodně vybraných reprezentantů z organizací různých velikostí s wiki softwarem.
2. Na základě identifikovaných problémů navrhnout low fidelity wireframy.
3. Navrhnout a implementovat webovou aplikaci, která se pokusí řešit identifikované problémy participantů.
4. Provedení uživatelského výzkumu na výsledné aplikaci a sepsání dalších možných vylepšení, které mohou představovat příležitosti pro další rozvoj v budoucnosti.
5. Prokázání funkčnosti aplikace otestováním několika vybraných scénářů vycházejících z případů užití.

Seznam doporučené literatury:

- [1] B. Leuf, W. Cunningham - The Wiki way, Addison-Wesley, 2001
- [2] J. West, M. West - Using Wikis for Online Collaboration, Jossey-Bass, 2009
- [3] T. Lowdermilk - User-Centered Design, O'Reilly Media, 2013
- [4] J. Gothelf - Lean UX, O'Reilly Media, 2013
- [5] O. Fernandez - The Rails 5 Way, Addison-Wesley Professional, 2017
- [6] M. Hartl - Ruby on Rails Tutorial, Addison-Wesley Professional, 2016
- [7] B. Frost - Atomic Design, Brad Frost, 2016

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Pavel Šedek, katedra ekonomiky, manažerství a humanitních věd FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **10.05.2019**

Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **19.02.2021**

Ing. Pavel Šedek  
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Ing. Pavel Ripka, CSc.  
podpis děkana(ky)

### III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.  
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

\_\_\_\_\_  
Datum převzetí zadání

\_\_\_\_\_  
Podpis studenta

## **Declaration**

I hereby declare that this thesis is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Marek Długoš

**Advisor:** Ing. Pavel Šedek



## Acknowledgements

I would like to thank my supervisors: Ing. Pavel Šedek from Czech Technical University as well as Ing. Carlos Fontela from University of Buenos Aires. I would like to thank Mgr. Tomáš Vestenický for mentoring me and showing me the best practices in the beginning of my journey with application framework Ruby on Rails. Lastly, let me thank Karla Losoya, BA, BSN, RN who helped with proofreading this thesis.

## **Abstract**

The wiki software has been on the market for quite a while. Schools, startups, nonprofits, and even enterprises quickly adopted it to share information across their organizations and collaboratively improve them. However, throughout the years, web standards have progressed very aggressively, bringing lots of new user requirements, architecture patterns and features that most of the current Wiki solutions simply cannot keep up with. This thesis presents a new wiki web application that aims to address these issues following User-Centered Design methods along with its implementation.



## **Keywords**

wiki, wiki software, content management system, knowledge base, knowledge management, collaboration, web application, user-centered design, design thinking, user research



# Contents

<b>Introduction and Motivation</b>	<b>1</b>
<b>1 A Wiki Software</b>	<b>3</b>
1.1 <i>History of Wiki Software</i> . . . . .	3
1.2 <i>Wiki Software Nowadays</i> . . . . .	4
<b>2 User-Centered Design</b>	<b>5</b>
2.1 <i>Design Thinking</i> . . . . .	5
2.2 <i>User Research</i> . . . . .	7
<b>3 Product Discovery</b>	<b>9</b>
3.1 <i>Competitive Analysis</i> . . . . .	9
3.2 <i>Personas</i> . . . . .	12
3.3 <i>Initial User Interviews</i> . . . . .	14
3.4 <i>User Research Results</i> . . . . .	14
3.5 <i>Scope Definition</i> . . . . .	16
3.5.1 <i>Must Have</i> . . . . .	17
3.5.2 <i>Should have</i> . . . . .	18
3.5.3 <i>Could have</i> . . . . .	20
3.5.4 <i>Won't have this time</i> . . . . .	22
<b>4 Low-Fidelity Wireframes</b>	<b>25</b>
<b>5 Technology Stack Decisions</b>	<b>29</b>
5.1 <i>Ruby and Ruby on Rails</i> . . . . .	29
5.2 <i>PostgreSQL</i> . . . . .	31
5.3 <i>ElasticSearch</i> . . . . .	32
5.4 <i>Sass</i> . . . . .	33
5.5 <i>CSS Framework</i> . . . . .	33
5.5.1 <i>Bootstrap and Bulma Comparison</i> . . . . .	34
<b>6 Implementation Decisions</b>	<b>37</b>
6.1 <i>MVC Architecture</i> . . . . .	37
6.2 <i>Multitenancy Architecture</i> . . . . .	38
6.3 <i>Database Design and Seed Data</i> . . . . .	38
6.4 <i>Polymorphic Associations</i> . . . . .	40
6.5 <i>Tree Structures</i> . . . . .	42
6.6 <i>Security</i> . . . . .	43

6.6.1	Authentication Solution . . . . .	43
6.6.2	User Roles Management . . . . .	44
6.6.3	Password Strength Estimation . . . . .	46
6.6.4	Passwords from Data Breaches . . . . .	47
6.7	<i>Search and Search Analytics</i> . . . . .	47
6.8	<i>Testing Framework</i> . . . . .	48
<b>7</b>	<b>Evaluation</b>	<b>49</b>
7.1	<i>User Interviews</i> . . . . .	49
7.2	<i>Software Testing</i> . . . . .	50
7.2.1	Feature Tests . . . . .	51
<b>8</b>	<b>Deployment</b>	<b>55</b>
<b>9</b>	<b>Future Development</b>	<b>57</b>
	<b>Summary</b>	<b>59</b>
<b>A</b>	<b>Initial User Interviews</b>	<b>61</b>
A.1	<i>Supporting Questions</i> . . . . .	61
A.1.1	About Participant . . . . .	61
A.1.2	Product Related Questions . . . . .	61
A.1.3	After showing 2 Wireframes . . . . .	62
A.2	<i>Interviews Transcript</i> . . . . .	62
A.2.1	Participant no. 1 . . . . .	62
A.2.2	Participant no. 2 . . . . .	64
A.2.3	Participant no. 3 . . . . .	66
A.2.4	Participant no. 4 . . . . .	67
A.2.5	Participant no. 5 . . . . .	68
A.2.6	Participant no. 6 . . . . .	70
<b>B</b>	<b>Evaluation User Interviews</b>	<b>73</b>
B.1	<i>Supporting Questions</i> . . . . .	73
B.1.1	Product Related Questions . . . . .	73
B.2	<i>Interviews Transcript</i> . . . . .	73
B.2.1	Participant no. 1 . . . . .	73
B.2.2	Participant no. 2 . . . . .	74
B.2.3	Participant no. 3 . . . . .	75
B.2.4	Participant no. 4 . . . . .	76
B.2.5	Participant no. 5 . . . . .	76

<b>C Wireframes</b>	<b>79</b>
C.1 <i>Landing Page</i> . . . . .	79
C.2 <i>Onboarding</i> . . . . .	80
C.3 <i>Wiki</i> . . . . .	84
C.4 <i>Administration Interface</i> . . . . .	92
<b>D Source Code</b>	<b>97</b>
<b>Bibliography</b>	<b>99</b>



## Introduction and Motivation

Sharing the information within an organization efficiently used to be difficult. However, many organizations around the globe realized that Wiki Software can help them achieve this goal and, have already been using it for decades; schools, startups, nonprofits, and even enterprises. A piece of software that enables them to **create and collaboratively edit "pages" or entities** via a web browser[1].

The main benefits of this were basically making information that is addressed to a bigger group of people within an organization **transparent, accessible and always up-to-date**. Organizations would use Wiki software to create their knowledge bases (single source of truth), help centers, checklists, documentation et cetera.

However, throughout the years, web standards have progressed very aggressively, bringing lots of new user requirements (like accessibility from mobile devices), architecture patterns (e.g. reactive UIs) and features that **most of the current Wiki solutions simply cannot keep up with**.

Remember good old discussion forums based on phpBB<sup>1</sup> or blogs build in Wordpress<sup>2</sup>? Well, to phpBB there comes a new alternative called Discourse<sup>3</sup> and to Wordpress there is Ghost platform<sup>4</sup>. Both are brand new systems meeting all the modern web application user requirements. But what about the wiki software? Is there any newcomer to the game? None I would know of. And this is why I have decided to tackle this problem.

I gave myself a task to apply methods of User-Centered Design to **develop a brand new Wiki software** written in modern technologies, attracting the modern user. That meant to conduct a user research; recruit and interview people who use Wiki software within their organizations, find problematic points and identify issues with current solutions. Provide a competitive analysis and create low-fidelity wireframes. Write an application that represents a working prototype, iterate on identified problems and validate the solutions with the group of participants.

---

1. <https://www.phpbb.com/>

2. <https://wordpress.com/>

3. <https://www.discourse.org/>

4. <https://github.com/tryghost/ghost>





# 1 A Wiki Software

A wiki is *"web-based software that allows all viewers of a page to change the content by editing the page online in a browser. This makes the wiki a simple and easy-to-use platform for cooperative work on texts and hypertexts."*[1]

Wikis were designed for groups to create, share, and collaborate on the content at any time and from whatever place. Wikis should be more efficient than forwarding e-mails, versioning its attachments and eliminating conflicting versions of the same document thanks to the group writing and group editing. Documents that are placed in the wiki are available for editing and commenting to all members at all times. No one has to wait and it is easy to see who contributed with what content. The only thing that a member of the wiki needs is a web browser. Then, based on the access and permission settings, he or she can create pages and/or edit existing pages, comment, etc..[2]

## 1.1 History of Wiki Software

The first ever wiki software is considered to be the WikiWikiWeb. Ward Cunningham started developing the WikiWikiWeb<sup>1</sup> in 1994 as an automated supplement to the Portland Pattern Repository<sup>2</sup>, a documentation about Design Patterns. The project aimed to become a collaborative database, to allow programmers to exchange their ideas easier. WikiWikiWeb was developed by Cunningham in Perl programming language. The site became popular within the pattern community immediately.[3]

The WikiWikiWeb was evolving, getting more and more features while others, similar projects have started popping up.

Since this idea started, and its primary focus was around the programming community, the concept was not popular with the public. This has changed after the success of the Wikipedia<sup>3</sup>, and organizations began to make increasing use of wikis.

We can consider wikis as publicly accepted with the word "wiki" entering the Oxford English Dictionary<sup>4</sup> in March 2007.

---

1. <http://wiki.c2.com/>

2. <http://wiki.c2.com/?PortlandPatternRepository>

3. <https://www.wikipedia.org/>

4. [https://www.theregister.co.uk/2007/03/16/wiki\\_oed/](https://www.theregister.co.uk/2007/03/16/wiki_oed/)

### 1.2 Wiki Software Nowadays

Probably the most popular Wiki nowadays is Wikipedia<sup>3</sup>. Wikipedia is *"free online encyclopedia with completely open content: nearly every article can be edited by anyone. Since its introduction in 2001, Wikipedia has grown to be the most popular general reference work on the Web."*[2]

Wikipedia is public; opened to everyone on the internet. However, I, myself, have never come across an organization with five and more people that would not use some kind of centralized place to store and share the knowledge. It is nearly impossible to count the number of organizations using wiki software but, when considering one of the largest open source wiki projects MediaWiki (which is also a backbone for Wikipedia) and their Usage Report<sup>5</sup> from 2015, there have been over 102,000 downloads of MediaWiki in about a month. These downloads came mostly from China, the United States, and Germany. Indeed, not every single download represents a running instance of the MediaWiki in some particular organization but it can give the reader a clear picture of the popularity of the wiki software.

These days, the solutions that are available still do not fully focus on user experience and (in some cases) does not take advantage of the available modern web technologies. Further documentation can be found in the Competitive Analysis section on page 9.

With an increasing number of formats in which we store the information, other solutions have popped up and are now used as wikis, too. To better illustrate it, it is worth mentioning G Suite (Google Documents), Dropbox, Microsoft OneDrive,... Even though these products come with many features and are developed by many experienced engineers, when used as a wiki software it is harder to implement community and social features.

After considering all of this information, I have realized that there might still be space for a wiki solution that would fill the gap on the market and improve this area.

---

5. [https://www.mediawiki.org/wiki/MediaWiki\\_Usage\\_Report\\_2015#Downloads\\_and\\_Hostings](https://www.mediawiki.org/wiki/MediaWiki_Usage_Report_2015#Downloads_and_Hostings)

## 2 User-Centered Design

Many people have resources and access to develop their own applications today. New products would appear on the market every single day. Not so many of them though would start with the problem statement. Many creators would envision the solution themselves and they try to find a group of people who would use their product. This fact causes a lot of newly published software to be rarely used as it was intended to. Usually, people would either not find the value to use this software or worst-case scenario the software would be even hard to use.

In my work, I have decided to take a reverse approach to this false process many makers begin with and start with the actual users who would utilize the software. This process is usually referred to as User-Centered Design and Travis Lowdermilk defines it as: *"a methodology used by developers and designers to ensure they're creating products that meet users' needs."*[4]

An important part of User-Centered Design is working with people who are intended to use the software. User-Centered Design is *"a mindset that overlays design thinking to ensure that the products are actually relevant and beneficial."*[5] Both, User-Centered Design as well as Design Thinking have an empathize/research phase and test phase in which methods used in user research come in handy.

### 2.1 Design Thinking

Design thinking, according to Tim Brown, CEO and president of world renowned design company IDEO is: *"innovation powered by...direct observation of what people want and need in their lives and what they like or dislike about the way particular products are made, packaged, marketed, sold, and supported...[It's] a discipline that uses the designer's sensibility and methods to match people's needs with what is technologically feasible and what a viable business strategy can convert into customer value and market opportunity."*[6]

Hence, I have started with the empathize phase (see all the phases in the Figure 2.1 on the next page) and conducted the research consisting of competitive analysis and initial interviews to better understand the needs of people using the wiki software.

The results of the initial interviews are covered in the User Research Results section summarizing the people's problems (define phase). Moving on to ideate phase, I have defined the scope of my work and prototyped the selected features in code (prototype phase).

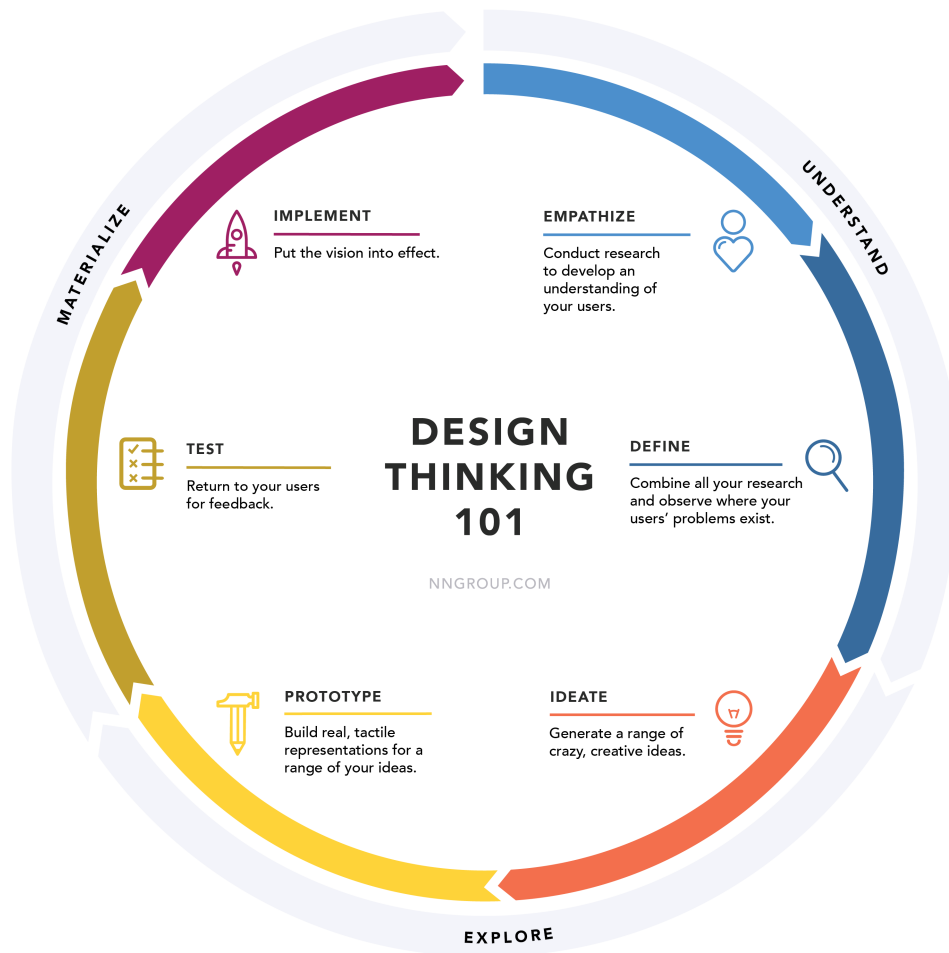


Figure 2.1: Design Thinking process, from [7].

Lastly, I have put the built and fully working prototype in front of the small group of carefully selected participants (test phase).

## 2.2 User Research

User research is about understanding user behaviors, needs, and motivations through observation techniques, task analysis, and other feedback methodologies.[8]

The user research techniques I have used in my work are:

- Competitive Analysis.
- Recruiting and Interviewing.
- Personas (in certain sense).
- Wireframing.
- Prototyping.
- Heuristic Evaluation.



## 3 Product Discovery

### 3.1 Competitive Analysis

Since the idea for this application appeared in my head, I continuously gathered other software that tried to solve either the same or a similar problem. Some were reviewed earlier and motivated me to work harder while some that were reviewed later were very similar to what I worked on.

It is important to point out that there are many wiki solutions out there if not hundreds and everyone can rate their strengths and weaknesses in their own ways. It is hard to cover all the wiki solutions and all the pros and cons. I used my *subjective optic* and vision that I have incorporated in the new solution to compare the various solutions that I came across. Another noticeable point to mention is that most of the software projects are *constantly changing and evolving*. Hence, the pros and cons are relevant to a point of time in which I worked on this project but might not necessarily represent the future.

First, let me briefly mention characteristics that I want to achieve with my solution:

- Modern, clean, and responsive design.
- Modern technologies used.
- Data analytics, actionable dashboards, and statistics.
- Simple WYSIWYG editor that supports consistency of the content.
- Outperforming search.
- Easy deployment.

### 3. PRODUCT DISCOVERY

---

The following table lists the wiki software solutions I have been tracking down since December 2017, its pros and cons as described earlier.

Product	Pros	Cons
Confluence	Clean design. Popular among enterprises.	Does not work on hand held devices.
XWiki	Open source. Responsive.	Robust WYSIWYG editor allowing way too many styles. Need to know the specific syntax to write content.
MediaWiki	Open source.	Obsolete design. Harder to discover content.
DokuWiki	Open source.	Basic skin design is obsolete. Need to know the specific syntax to write content.
ZohoWiki	Modern design. Integration with other systems.	Robust WYSIWYG editor allowing way too many styles.
MindTouch	Basic dashboards.	Obsolete design.
Sabio	Basic suggestions based on data. Basic analytics.	Slightly obsolete design. Robust WYSIWYG editor allowing way too many styles.
BrainKeeper		Obsolete design. Obsolete technology.
Wikisystems		Obsolete design. Obsolete technology.



TWiki	Open source.	Obsolete design. Obsolete technology. Robust WYSIWYG editor allowing way too many styles. Need to know the specific syntax to write content. Non-actionable stats.
WikkaWiki	Open source.	Obsolete design. Obsolete technology. Weak usability.
PmWiki	Open source.	Obsolete design. Obsolete technology. Very specific design.
Wikispaces	In-context comments.	Non actionable stats.
Wikidot	Gamificaiton.	Obsolete design.
PB Works Wiki		Obsolete design. Obsolete technology. Obsolete editor.
Inmagic Presto	Responsive	Cluttered design. Poor usability.

Table 3.1: Subjective pros and cons of various wiki softwares.

Some other solutions that do not necessarily call themselves wiki solutions but might represent a strong competition in this field are:

- **Google Drive** - Provides powerful abilities to create documents, share and search through them. However, lacks the whole wiki (community) environment.
- **Dropbox Paper** - Provides intuitive editor for content creation and sharing. Same as Google Drive misses the sense for wiki community.
- **Notion** - Powerful features for content creation and collaboration. Does not provide any dashboards.
- **Gitbook** - Comes with a powerful editor. No actionable statistics.
- **Slite** - Very similar tool to my target solution.
- **HelpSite** - A bit obsolete design. No actionable statistics.
- **You need a wiki** - Interesting extension to Google Drive that basically adds some basic wiki functionality to already existing Google documents.

## 3.2 Personas

Personas usually describe the main group of people that uses or will use the application.

An example of the persona using an IDE could look like: "John is a 30 years old single man, holding MSc. from Computer Science, working at a software company as a Software Engineer using mainly C programming language. He is very tech savvy and loves to write clean and easy to understand the code. One day he wants to lead a small team of developers."

One of the pillars of the whole user-centered design and design thinking is empathy. Personas help to create empathy. However, they can fail in the real world, because *"people who are categorized based on personas can have different jobs to be done, regardless of their age, occupation, location, status, education background and etc.."*[9]

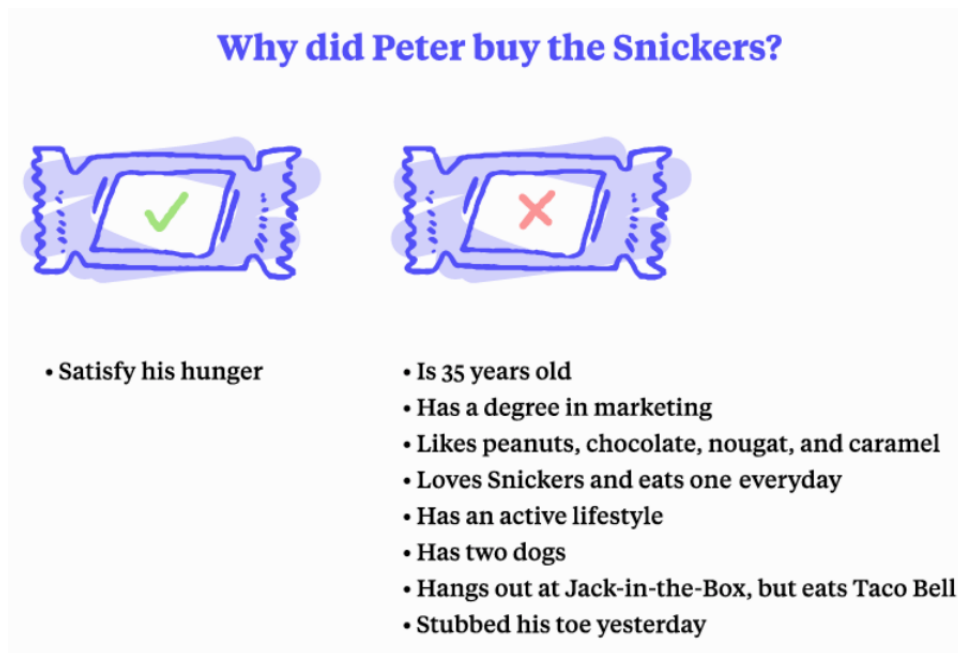


Figure 3.1: Why personas fail in real world, from [10].

Despite this fact, I have decided to at least briefly sketch who uses Wiki Software within organizations so it helps me not to define what they need but only create deeper empathy.

Organizations using Wiki Software usually have:

- **Regular members of the organization** - most of the time consuming the content.
- **Experienced members of the organization** - able to judge what is important and what is not. Able to contribute their knowledge.
- **Technical staff of the organization** - cares about onboarding new members, deals with technical issues.
- **Organization management** - makes the decision about purchasing and using certain software.

This sketch later led to the definition of certain user roles with certain user permissions in the application.

#### 3.3 Initial User Interviews

Following User-Centered Design, I conducted research on the type of people who would use this application upon completion and also provided feedback along the way during every phase with their feedback and insights from the industry they are in.

Therefore, I have carefully picked a small group of people across various organizations with a focus on diversity in the organization size as I believe that organizations that vary in size have different needs.

While small organization want to have something up and running for small costs, other big enterprises might require extra security, permission management, etc.. My group consists of people that vary from entry-level positions through C-level managers. I collected opinions across small startups with 1-30 people and one office to huge corporations having employed thousands of people across the globe, with offices in almost every major city.

First, I sat down and thought about what kind of information I wanted to get from people, what would help me build better software?

As a result, I ended up with a list of (as I call it) supportive questions which the reader might find in the Appendix A. Since interviews were unstructured; aimed to be as open as possible and every participant experience was different, sometimes the interview headed towards a more technical direction, other times, it went towards a totally business direction. At times, some interviewees proposed a totally different approach to sharing the information within their company. Most of the interviews were done in person but those that were not possible to get done in person were done through a video call. The interviews lasted typically anywhere from 30 minutes to 1 hour.

#### 3.4 User Research Results

Based on the competitive analysis I have done; I definitely see a big opportunity in *innovating the user interface* of the wiki software. The design looks obsolete and with the high growth of people using smartphones to consume information nowadays, current solutions hardly respond to their needs. Their user interfaces are broken or not even working on smaller devices. Other solutions are usually stuck with old fashioned, very complex WYSIWYG editors that enable people to write every page in a totally different format. I see this as a good thing for a person who is willing to dedicate two hours to formatting but I also see this as a great opportunity to bring a new writing experience to the wiki field. Where people would have a distraction-free

editor with basic formatting capabilities which would benefit to the consistency of the content across the whole wiki. All those problems have their roots also in the obsolete technology that might be modern back in the days but not now.

Also, during the user interviews, I have conducted, there were answers that repeated. The user interviews have shown that the most important elements in Wiki Software for the research participants are:

- Discoverability.
- Search.
- Rights management.

The biggest problems that the research participants are facing using Wiki Software are:

- Information become obsolete fast.
- Poor search.
- Weak information architecture and usability.

Some other requirements worth mentioning that the research participants need from a Wiki Software are:

- On-premise solution.
- Single sign-on (SSO).
- Easy deployment.

Hence, the biggest opportunities I have identified based on user research are:

- Working with data to motivate people to write new content.
- Suggest to people what to write next.
- Suggest people review potentially obsolete content to keep the wiki updated and compact.
- Tailor the recent activity feed so it's not spam.
- Try to overcome duplicates.
- Providing the pre-made document templates.
- ...and many other previously mentioned.

## 3.5 Scope Definition

When I started working on the product development, I quickly realized how easily the scope of the project could grow.

Based on the results of the conducted user research, I created a detailed plan of what the final product should include and what is fine to implement in near future based on MoSCoW Prioritization <sup>1</sup> and methodology called Jobs To be Done <sup>2</sup> popularized by company called Intercom <sup>3</sup>.

MosCoW Prioritization is *"a prioritisation technique for helping to understand and manage priorities."*[11]

The letters in the name stands for:

- Must Have
- Should Have
- Could Have
- Won't Have this time

While Jobs To be Done methodology describes use cases in the following format:

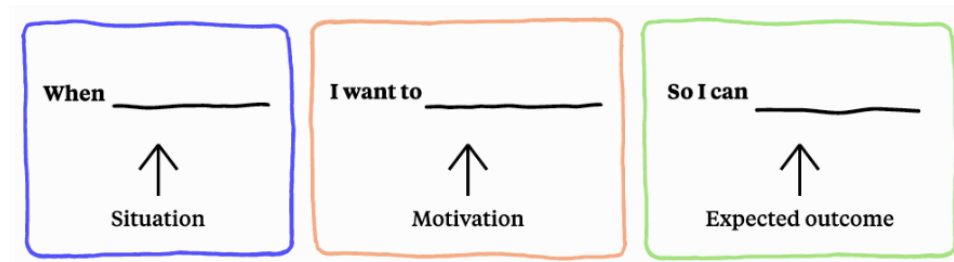


Figure 3.2: Jobs to be done formula, from [10].

Focusing on the triggering event or situation, the motivation and goal, and the expected outcome.

**Example:** When my thoughts are not polished, I want to save a page as a draft, so I can get back to it later while also keeping it concealed from other users.

The following sections list all the features I have either implemented or thought about implementing in the future.

---

1. <https://www.agilebusiness.org/content/moscow-prioritisation>

2. <https://www.intercom.com/books/jobs-to-be-done>

3. <https://www.intercom.com/>

### 3.5.1 Must Have

#### **MH001 - Create wiki** (✓ Implemented and tested)

When there is the information that repeats and need to be shared with everyone, I want to create wiki, so I can store the information.

#### **MH002 - Edit wiki** (✓ Implemented and tested)

When a name of the organization changes, I want to edit wiki, so I can better reflect on changes in the organization.

#### **MH003 - Delete wiki** (✓ Implemented and tested)

When the organization comes to its end, I want to delete the wiki, so I can make sure that all the information was deleted.

#### **MH004 - Create page** (✓ Implemented and tested)

When some questions across the organization repeat, I want to create a page, so I can share it with other members.

#### **MH005 - Edit page** (✓ Implemented and tested)

When I spot a chance for improvement, I want to edit a page, so I can keep the wiki up to date.

#### **MH006 - Delete page** (✓ Implemented and tested)

When the information on the page is obsolete or the page is not important anymore, I want to delete the page, so I can keep the wiki compact.

#### **MH007 - Create pages structure** (✓ Implemented and tested)

When one page closely relates to another one, I want to create some sort of structure between pages, so I can keep the wiki easy to navigate and discover.

#### **MH008 - Sign up** (✓ Implemented and tested)

When accessing the landing page, I want to sign up, so I can create any type of content.

### 3. PRODUCT DISCOVERY

---

#### **MH009 - Sign in** (✓ Implemented and tested)

When accessing the landing page, I want to sign in, so I can access my information.

#### **MH010 - Invite new member** (✓ Implemented and tested)

When there is a new member to organization, I want to invite him to the wiki, so I can share the information with him.

#### **MH011 - Edit user** (✓ Implemented and tested)

When there the information in my profile is obsolete, I want to edit my profile, so I can keep my profile up to date.

#### **MH012 - Delete user** (✓ Implemented and tested)

When a member of the organization leaves, I want to be able to delete his account, so I can keep only the members of the organization as members of the wiki.

#### **MH013 - Fulltext search** (✓ Implemented and tested)

When I look for some information, I want to be able to search, so I can retrieve the information faster.

#### **MH014 - WYSIWYG editor** (✓ Implemented and tested)

When I write a new page, I want to be able to format it easily, so I can create a better visual structure of the page.

### 3.5.2 Should have

#### **SH001 - Multitenancy** (✓ Implemented and tested)

When I create a new wiki, I want to separate the information in the database, so I can ensure better security, scalability, and own subdomain.

#### **SH002 - Save page as a draft** (✓ Implemented and tested)

When my thoughts are not polished, I want to save a page as a draft, so I can get back to it later while also keeping it concealed from other people.



**SH003 - Publish page** (✓ Implemented and tested)

When my thoughts are complete, I want to publish a page, so I can share it with my colleagues.

**SH004 - Archive page** (✓ Implemented and tested)

When the page is obsolete, I want to archive it (make it invisible to other members), so I can keep the wiki compact but still have a chance to get back to the archived page.

**SH005 - Browse previous versions** (✓ Implemented and tested)

When the page contained useful information previously, I want to be able to get back to the previous version, so I can read this information.

**SH006 - Show diff of the version** (✓ Implemented and tested)

When there were changes made to the page, I want to see what has been added or removed, so I can quickly get an idea of what changes were made.

**SH007 - Rollback previous version** (✓ Implemented and tested)

When the page contained useful information in previous version, I want to rollback that version, so I can retrieve the useful information.

**SH008 - Comment on page** (✓ Implemented and tested)

When there is something unclear on a page, I want to comment on a page, so I can notify an author or spark a discussion.

**SH009 - Reply on comment** (✓ Implemented and tested)

When the comment contains a question, I want to be able to reply on comment, so I can answer that question.

**SH010 - Delete comment** (✓ Implemented and tested)

When the comment contains false information, I want to be able to delete it, so I can stop other members from reading it.

**SH011 - Recent Activity** (✓ Implemented and tested)

When someone posts or edits something, I want to know about it, so I can stay up to date with what is going on.

#### **SH013 - User Roles and Permissions** (✓ Implemented and tested)

When there is a new member to wiki, I want to set him a role, so I can assign him different permissions within the wiki.

#### **3.5.3 Could have**

##### **CH001 - React on page** (✓ Implemented and tested)

When the content of the page is engaging, I want to be able to react to the page, so I can express my opinion or emotions.

##### **CH002 - See who reacted** (✓ Implemented and tested)

When the page is engaging, I want to see who reacted and how, so I can better understand people's emotions or opinions about the content of the page.

##### **CH003 - Cancel the reaction** (✓ Implemented and tested)

When I used undesired reaction, I want to be able to take back my reaction, so I do not confuse other members of the wiki.

##### **CH004 - Pin page** (✓ Implemented and tested)

When I use some page often, I want to be able pin it, so I can leverage the knowledge faster.

##### **CH005 - Unpin page** (✓ Implemented and tested)

When I stop visiting a page often, I want to be able to unpin it, so I can keep my dashboard clean.

##### **CH006 - Password strength estimation** (✓ Implemented and tested)

When signing up, I want to see how strong is the password I typed, so I am sure I use strong password.

##### **CH007 - Check password against PwnedPassword** (✓ Implemented and tested)

When signing up, I want to check the password against the PwnedPassword dataset, so I am sure that password is safe to use.

**CH008 - Own profile picture** (✓ Implemented and tested)

When I am familiar with software, I want to upload a profile picture, so I can allow other people to quickly recognize me.

**CH009 - Gravatar profile picture** (✓ Implemented and tested)

When there is no profile picture uploaded, I want to use Gravatar picture, so I can give a new member more "personality".

**CH010 - Search terms with no results** (✓ Implemented and tested)

When there is term searched often with no results, I want to write a new page on this topic, so I can help other members.

**CH011 - Most searched terms** (✓ Implemented and tested)

When looking for the information, I want to see the most searched terms, so I can see if this information was not already been searched by other members of the wiki.

**CH012 - Least viewed pages** (✓ Implemented and tested)

When thinking of obsolete content that could be archived, I want to be able to display the least viewed pages, so I can judge if they should remain the part of the wiki.

**CH013 - Poorly rated pages** (✓ Implemented and tested)

When some page has been ranked as not helpful, I want to be able to see it, so I can review the page.

**CH014 - Tests** (✓ Implemented and tested)

When any functionality in the application is broken, I want to be notified, so I can fix it.

**CH015 - Seed data** (✓ Implemented and tested)

When I newly install the application, I want to have some data in my database, so I can test the application right away.

#### 3.5.4 Won't have this time

##### **WH001 - Autocomplete**

When searching for something, I want the application to suggest me the complete query, so I can speed up my process of searching.

##### **WH002 - Suggestions**

When searched for something that is not absolutely correct, I want the application to suggest me a correction, so I can always find what I am looking for (of course, if it is available).

##### **WH003 - Single Sign-On**

When using a company account, I want to be able to use single sign-on, so I can avoid duplicate accounts.

##### **WH004 - 2FA Authentication**

When signing in, I want to verify my identity through some other device, so I can ensure my account is always secure.

##### **WH005 - Edit comment**

When commented something that can be revised, I want to edit my comment, so I can avoid spamming the comment thread.

##### **WH006 - React on comment**

When agreeing or disagreeing with a comment, I want to react on comment, so I can express my emotion or opinion.

##### **WH007 - Watch page**

When page contains important and often changing information, I want to be notified on changes, so I can always get up to date information.

##### **WH008 - Watch discussion**

When there is a important discussion, I want to be notified on new comments, so I can stay up to date with what is going on.

**WH009 - Email digests**

When there are news to the wiki, I want to send out an email to the members, so I can notify them about new content.

**WH010 - Rate helpfulness of page**

When I finish reading a page, I want to rate whether it was useful for me or not (and why), so I can give the author of a page feedback.

**WH011 - Show messages based on updated\_at**

When displaying page that has not been updated for a long time, I want to display a warning about this fact, so I can be aware that the information in the page might not be up to date.

**WH012 - Real time collaboration**

When editing or creating a page, I want to write or edit the content real time with other people, so I can better collaborate.

**WH013 - Create page based on template**

When creating a new page, I want to use a pre-saved template, so I can keep the format of pages consistent or follow best practices of my organization.

**WH014 - Manage page templates**

When creating best practices for pages (or format), I want to manage page templates, so I can set best practices for creating pages for other members.

**WH015 - Review changes by other members**

When another person with lower permissions makes changes, I want to review these changes, so I can avoid publishing information that I do not approve of.

**WH016 - In-context comments**

When I spot a mistake or see room for improvement, I want to comment on that page, so I can easily refer to that specific section of the page.

#### **WH017 - Avoid duplicate pages**

When creating a new page, I want to see if similar pages already exist, so I can avoid the duplicate pages.

#### **WH018 - Breadcrumbs**

When I scan through page, looking for an information, I want to see hyperlinks to other pages in the tree, so I can navigate through pages with ease.

#### **WH019 - Feature flags**

When my organization does not need a certain feature, I want to disable this feature, so I can enjoy cleaner user interface.

#### **WH020 - Localizations**

When my organization uses languages other than English, I want to change the language, so I can make it easier for others to use the application.

#### **WH021 - On-premise solution**

When policies of my organization requires it, I want to use the application on my own server, so I can ensure that my data is in my control.

#### **WH022 - Onboarding emails**

When a new member signs up, I want to send him couple of automated emails, so I can show him how to get the most out of the application.

In total, more than **42 features have been implemented and tested** while 22 more features are in the backlog.

## 4 Low-Fidelity Wireframes

Since I have seen the big opportunity to innovate in design, I tried to follow the best practices design process and user-centered design methods. To iterate in the design process one has to start small. From pen and paper sketches, through low-fidelity wireframes made digitally, through interactive prototypes, through high fidelity designs, to coding the solution.

Testing of the output should occur at the end of each phase. That is, of course, the ideal scenario. In real life, many forward-thinking companies experiment with features on the a small percentage of people [12] using their software. I have decided to conduct user interviews at the beginning and then discuss the solution I have been working on at the end.

When picking a tool that would serve these needs, I focused on having one software which would cover all of the design phases rather than having multiple software with file formats for each phase.

I ended up picking Sketch<sup>1</sup>. The most used design software for designing user interfaces these days<sup>2</sup>, leveraged by biggest organizations in the world. It also integrates well with other modern design tools out there that I will use in future design phases for prototyping.

Despite Adobe Photoshop or other software that was used in the past, the Sketch software is perfectly aligned with designer needs and provides big efficiency and speed.

It also evolves very fast, and at the time when I was working on the implementation of the application, Sketch had already implemented some prototyping and sharing features. On top of that, Sketch allows for the creation of symbols. These are features that one can imagine as components - which is also one of the very modern architecture patterns on the front end. Designers would be able to create a very modular system (almost like it was coded). In the past, when for example top navigation had changed, a designer would have to go to the screen over a screen to apply those changes. With Sketch, I only change the symbol and all of the symbol instances gets updated immediately on all application screens.

---

1. <https://www.sketch.com/>

2. <https://avocode.com/design-report-2018>

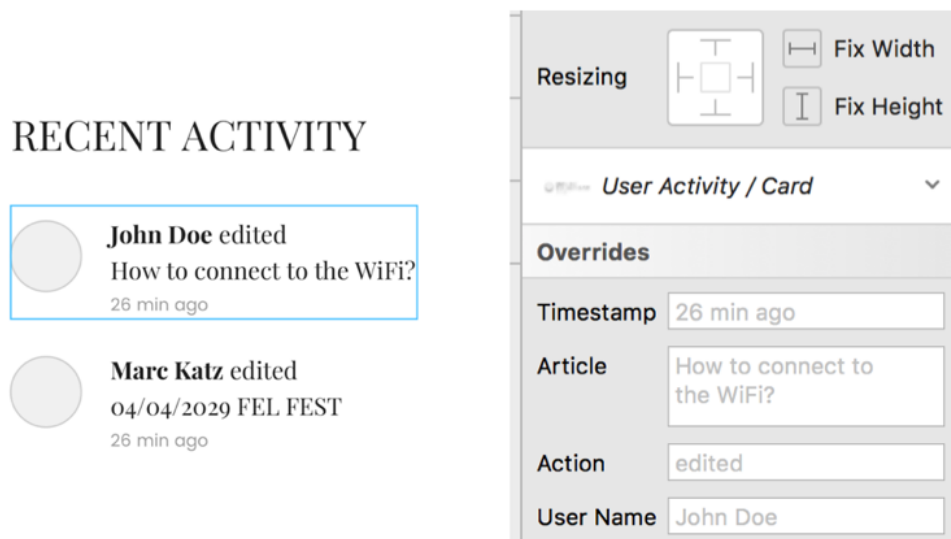


Figure 4.1: An example of symbol overrides.

So I took advantage of this feature and created a small, tailored to my need wireframing library. It's modular and easy to pick up.

Following the user-centered approach, it also enables me to prototype to be able to present the outputs in an effective way and move quickly to ensure I can get the application public as soon as possible.



## 4. LOW-FIDELITY WIREFRAMES

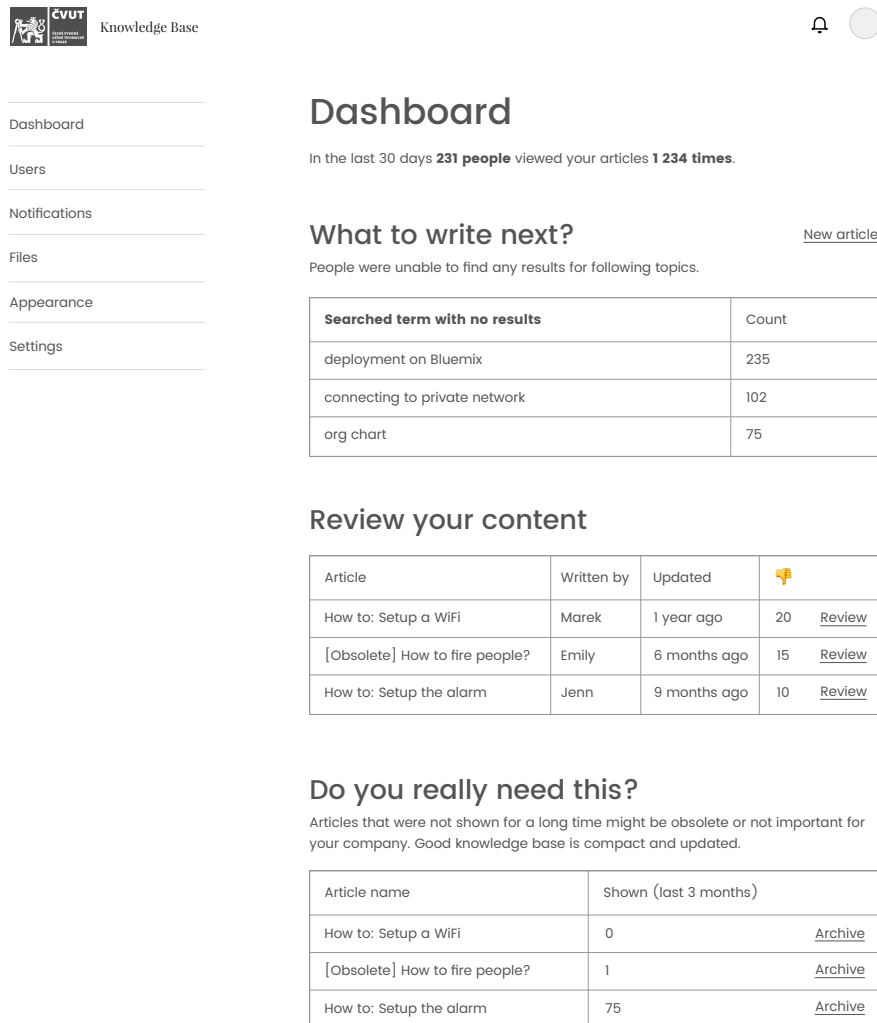


Figure 4.2: Example of an wiki dashboard.

See Appendix C for other low-fidelity wireframes I have produced.



## 5 Technology Stack Decisions

As with every new software, there are decisions to make first. It is up to the maker of the product to decide which way he or she wants to go. Because the title of my work contains the word "modern", I have decided to focus on the latest versions of technologies that I found interesting and suitable for implementation.

### 5.1 Ruby and Ruby on Rails

Because Wikis are supposed to be available to every member with only internet browser it was obvious that the application would have to run on the web. To speed up the workflow and "not reinvent the wheel" I have decided to support my implementation with a web framework. There are only a couple of dominant web frameworks these days, I consider worth mentioning. Some of them are Express (written in JavaScript), Django (written in Python), Ruby on Rails (written in Ruby), Laravel (written in PHP) and Spring (written in Java).

Another goal I have decided to accomplish was to learn a new programming language and at the same time take into consideration the time frame that was given to me to finish this project.

While all the frameworks are very similar each of them features something slightly different. For instance, Express benefits from the powerful performance of the asynchronous Node.js, however, there's no defined way of doing things, at least for beginners. Laravel, Django, and Spring are all popular frameworks with big communities around them. But, because I have had the pleasure to write some PHP, Python, and Java before, for this project, I decided to explore, to me, unknown waters of Ruby and it is a most popular web framework Ruby on Rails.

Another factor that played a role in this decision is that Ruby on Rails has been known for a long time as a web framework for rapid development.[13] What made it famous for rapid development was the introduction video first published by the creator of Ruby on Rails where he was able to get a basic web blog working from scratch in less than twenty minutes. So given the time frame to finish this work, Ruby on Rails represented the best option to go for.

Ruby on Rails is used by companies like Airbnb, Disney, Github, Kickstarter, Shopify, and Twitter[14]. The framework is 100% open-source, available under the MIT license which allows not only download and use the

framework but also improve it and contribute to it. And who contributes to this framework? Well, another fact for which Ruby on Rails is known is *"unusually enthusiastic and supportive community."*[15] This community which spans more than 5,300 contributors<sup>1</sup> (to Apr 10, 2019) results in plenty of tutorials that have been created, conferences that are organized on a regular basis and last but not least in a huge number of gems (*"self-contained solutions to specific problems such as pagination and image upload"*[15]) which is an advantage that I will use in my work as well.

When we talk about Ruby on Rails we talk about model-view-controller or in other words MVC framework, which separates an application into three interconnected parts. This design pattern allows (except other advantages) for efficient code reuse and parallel development. I describe this architecture pattern more in the MVC Architecture section.

Ruby on Rails framework allows me to work flexibly and iterate fast as I would pivot with business changes mainly due to the following features of the framework:

- **Do not Repeat Yourself - DRY** — *"DRY is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." By not writing the same information over and over again, our code is more maintainable, more extensible, and less buggy."*[16]
- **Convention over configuration - CoC** — *Conventions lower the barrier of entry for onboarding and allow mainly new team members to cooperate more flexibly, since many architectural decisions were already been made by the framework itself, which means many aspects of a Rails application are same across every Rails project.*[16]
- **ActiveRecord – ORM** — As my business logic changes constantly, so does the entities. ActiveRecord *"maps one domain class to one database table, and one instance of that class to each row of that database."*[17] It *"includes mechanisms for representing models and their relationships, CRUD (create, read, update and delete) operations, complex searches, validations, callbacks, and many more features."*[17]
- **Full stack framework** — Ruby on Rails serves to build a monolith application.[18] However, front-end technologies are widely supported and Rails 5 comes with Webpack support[19], which pays off in the long run.

---

1. <https://contributors.rubyonrails.org/>

## 5.2 PostgreSQL

Web framework Ruby on Rails allows us many options in terms of choosing the right database to store the application data. From a simple SQLite through MySQL, MongoDB to PostgreSQL. Before talking about the actual decision to choose PostgreSQL it is worth mentioning that Ruby on Rails allows us to change the database at any time with only a few lines of code.

Listing 5.1: Changing the database with a single line of code.

```
# database.yml
default: &default
  adapter: postgresql
  encoding: unicode
```

That is also thanks to migrations which are simple and unique atomic actions that are performed on the database.

Listing 5.2: Migration file for creating the comments table.

```
# 20190120042516_create_comments.rb
class CreateComments < ActiveRecord::Migration
  [5.2]
  def change
    create_table :comments do |t|
      t.integer :author_id
      t.text :body
      t.integer :commentable_id
      t.string :commentable_type

      t.timestamps
    end
  end
end
```

Migrations are written in Ruby language so there is no need to worry about precise SQL syntax. In other words, Ruby on Rails is *database agnostic*. The database is already abstracted for us. The final structure of the database is stored in the db/schema.rb file.

Listing 5.3: Part of the `schema.rb` file showing the creation of the comments table.

```
# schema.rb
...
create_table "comments", force: :cascade do |t|
  t.integer "user_id"
  t.text "body"
  t.integer "commentable_id"
  t.string "commentable_type"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
end
...
```

SQLite is primarily meant for development or small projects. It has various technical limitations with regards to performance, scalability, and others.

From the databases that were at my disposal, MySQL and PostgreSQL both looked like a good choice. In the past, we would find more important differences between those two but nowadays, one can not go wrong in choosing either of these two.

MySQL does not allow for the usage of full SQL compliance (ie. lacks support for FULL JOIN clauses) and is not fully open sourced. Some of its editions are released under proprietary licenses. PostgreSQL supports more of the full SQL compliance features with a long list of extensions, is fully open sourced and extensible, however, has bigger limitations on memory.[20]

Because I have worked with PostgreSQL before and never encountered any problem, I have decided to stick with this database in this project, too. As a reminder of what I have already said, it is up to the specific implementation or organization on what database they will decide to use and Ruby on Rails would make this switch basically seamless.

### 5.3 ElasticSearch

One of the options I had to implement the search functionality was to use the built-in capability of the database to do full text.

However, based on the results of the user research I have performed, it was clear that the requirements for search functionality are very high.

The search function turned out to be one of the most important elements in any wiki for participants. Hence, I wanted to make sure that the search

function that I implemented in the application would easily manage all kinds of requests people can make.

To achieve this goal, I decided on implementing the Elasticsearch search engine.<sup>2</sup> It not only provides a full-text search, but also learns what people are looking for. The more people search, the smarter it gets and the results are better.

Some of the features worth mentioning are: [21]

- Handling:
  - stemming - tomatoes matches tomato.
  - special characters - jalapeno matches jalapeño.
  - extra whitespace - dishwasher matches dish washer.
  - misspellings - zuchini matches zucchini.
  - custom synonyms - qtip matches cotton swab.
- Reindex with no downtime.
- Easily personalized results for each user.
- Autocomplete.
- “Did you mean” suggestions.

## 5.4 Sass

Sass is a CSS pre-processor which allows one to use features that do not exist in CSS yet like user variables, mixins, nesting and other tweaks to write modular and complex cascading styles.

Basically, it means that I would write seemingly more complicated CSS code, which I would save into a file with .scss extension and then would use Sass to compile this file and return a plain .css file as an output which would be used for the browser.

This decision not only speeded up my workflow in the beginning but also helped to keep the code easily maintainable in Future Development.

## 5.5 CSS Framework

Designing a consistent front end across multiple browsers, versions, and devices is hard. Something that could help me accomplish this task easier was a

---

2. <https://www.elastic.co/>

CSS Framework. They provide a rich list of predefined classes and rules that speeds up the workflow of designing the web applications tremendously.

According to the size of the community the top three CSS Frameworks are: Bootstrap<sup>3</sup>, Materialize<sup>4</sup>, and Bulma<sup>5</sup> in the respected order.

One of the factors in making the decision was to make the new user interface feel different and fresh while also trying to avoid people's responses like "This looks totally like a system in my Android phone." This was exactly the issue with Materialize CSS Framework which is based on the Google Material Design<sup>6</sup>. The same would apply to Bootstrap as it is the most widely used CSS Framework out there and is used among many of the new applications that were designed with limited resources. From this point of view, I have seen Bulma CSS Framework as a solution which still shares the characteristics of the big community around this project and at the same time as a solution that no other application that I have ever seen uses. Simply put, it felt different and yet fresh to me.

### 5.5.1 Bootstrap and Bulma Comparison

#### Bulma

- Using latest CSS3 features. Bulma aims to stay on the bleeding edge of browser technology.
- Easy to learn syntax.
- Simple grid system.
- No JavaScript. Bulma provides a lightweight solution that can easily be implemented in any development context.

#### Bootstrap

- Uses jQuery and includes some useful plugins to add interactivity.
- Larger community.
- Slightly better compatibility with IE11.
- Bootstrap has strong and pervasive compatibility with WCAG 2.0 guidelines.

---

3. <https://getbootstrap.com/>

4. <https://materializecss.com/>

5. <https://bulma.io/>

6. <https://material.io/design/>



Above comparison was paraphrased from Bulma's official website [22].

Since the name of my work includes the word "modern", I gravitated towards the concept of the bleeding edge browser that Bulma provides. The presence of jQuery in the project is widely discussed on the internet<sup>7</sup>. Some argue that it speeds up the workflow while others argue that many projects do not need jQuery at all for the basic DOM manipulation. Because my plans for Future Development (for more information continue reading this chapter) relies on strictly separating the front end from the back end, I decided to go for a leaner and lightweight implementation. That is another reason why I have chosen Bulma and have decided to write a pure vanilla JavaScript.

---

7. Some examples:

<https://hackernoon.com/you-truly-dont-need-jquery-5f2132b32dd1>

<https://blog.garstasio.com/you-dont-need-jquery/>

<http://youmightnotneedjquery.com/>

<https://github.com/nefe/You-Dont-Need-jQuery>



## 6 Implementation Decisions

### 6.1 MVC Architecture

MVC Architecture or also known as Model-View-Controller architecture is an architectural pattern that divides an application into three interconnected parts:

- **Model** - represents the shape of the data and business logic.
- **View** - is a user interface. View displays data and allows its modification.
- **Controller** - handles user requests.

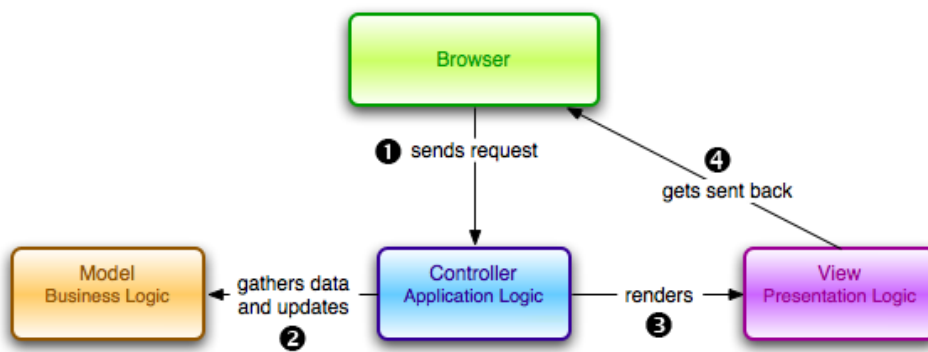


Figure 6.1: MVC Architecture, from [23].

Advantages of this approach are: [23]

- Better scalability.
- Logical grouping.
- Ease of maintenance.
- Reusability.

The MVC architecture has become popular for designing web applications and Ruby on Rails web framework is no exception. It follows this design pattern, however, at the same time allows for replacing the view component with simple JSON that allows for using the application as a REST

API. That is definitely something I would like to take advantage of in the Future Development.

More on MVC Architecture in Rails can be found in Sitepoint's article [23].

### 6.2 Multitenancy Architecture

When we talk about multitenancy we talk about *"a software development architecture approach in which each client gets their own app configuration and data."* [24] These can be isolated softly or strictly from other clients. Each "instance" is called a "tenant." [24]

Choosing this architecture from the beginning benefited the software from a few aspects. The first benefit is that it was built as a cloud based application. After deployment, one application is able to serve multiple organizations. This leads to lower costs of maintenance and updates, allow for better scalability, efficiency in performance, and convenient onboarding of new tenants (or in other words, people who create new wikis).

In my case, using Ruby on Rails and PostgreSQL, each tenant is a *separate PostgreSQL schema*. To allow more personalized feeling for people using the wiki, I have decided to specify wikis by *subdomains*. That means if the application as a whole will be located on `www.wik.com` then each organization using this application would be accessible through the subdomain like `www.organization.wik.com`. Accessing this subdomain would serve data from the "organization" database schema. However, some models would be better shared across all tenants. These are 'User', 'Wiki' and 'WikiUser' (representing the relationship between users, wikis, and their assigned roles on each wiki).

### 6.3 Database Design and Seed Data

The more features I was adding the more complex the database design was becoming. The following picture shows the entity relationship diagram of the actual database:



To fill the fresh installation of the application with some meaningful content fast and easy, I have created a set of basic dummy data with which the database could be populated and later tested.

These include creating new users with different roles on different wikis, and creating a bunch of pages and comments to them.

Listing 6.1: Showcase of seeding the database.

```
# seeds.rb
...
reader_ctu = User.find_or_create_by!(
  first_name: "Marek",
  last_name: "Dlugos",
  email: "reader@ctu.cz",
) do |user|
  user.password = "hlavneheslo"
  user.password_confirmation = "hlavneheslo"
end
...
ctu = Wiki.find_or_create_by!(name: "CTU",
  subdomain: "ctu")
...
```

When seeding the database, each entity would by default have an object which is fully populated with data, partially populated or very sparsely populated. That way it was easier to uncover and test out more scenarios.

### 6.4 Polymorphic Associations

Polymorphic association represents a relationship from one class to multiple classes. In other words, a model can belong to more than one other model, on a single association.

This came in handy in two specific implementations; nested comments and reaction on both, pages and comments.

Since one of my priorities was to encourage the social factor in the app, I wanted to provide people with as many opportunities to express their opinions and moods as possible. This resulted in an ability to spark the discussion below every page, and on top of that people can discuss in a better, more structured way thanks to the nested comments.

## Discussion

Add a comment

Post

G

**Martin Vanda** · 14 days ago · [Delete](#) · [Reply](#)

More to come!

G

**Daniel Novak** · less than a minute ago · [Delete](#) · [Reply](#)

Can't wait for more!

G

**Oliver Habala** · 13 days ago · [Delete](#) · [Reply](#)

Love this Prague office!

I am glad to hear that, Oliver!

Reply

Figure 6.3: Nested comments under an article.

Another opportunity to express the moods, agreeing or disagreeing with the certain page or comment is via reactions. These are little buttons attached to pages and comments in a form of Emojis, *"a visual representation of an emotion, object or symbol."*[25] Other members of the wiki can later see who reacted and how. This might come in handy, for example, with internal company updates, voting for the best option or solution in the comments, etc..

## Reactions

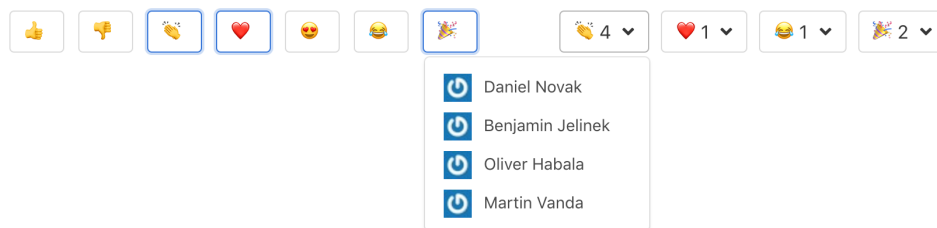


Figure 6.4: Reactions section under an article.

The above-mentioned features implementation would later look like this: A comment can belong to a page (or article in other words) as well as another comment (child comment would belong to the parent comment). The same applies to reactions. A reaction would belong either to a page or to comment.

Example of how I accomplished polymorphic associations with Rails in my work:

Listing 6.2: Polymorphic association in Ruby on Rails.

```
# comment.rb
class Comment < ApplicationRecord
  ...
  belongs_to :commentable, polymorphic: true
  has_many :comments, as: :commentable, dependent:
    :destroy
  ...
end
```

The model for reactions would then look like this:

- **id** - represents the ID of the reaction.
- **reaction\_type** - defines the type of the emoji (e.g. thumbsup, thumbsdown).
- **reactionable\_type** - this is where the class the reaction belongs to would be defined (e.g. Comment).
- **reactionable\_id** - the ID of the object that the reaction belongs to (e.g. 14, but in this example it would be a comment with ID 14).
- **user\_id** - who reacted.

Choosing this implementation I was able to better structure and reuse the code and avoid extra database tables.

### 6.5 Tree Structures

Along the way of implementation, there have been cases where the real world interpretation reflected tree structures, hence, I went ahead and implemented this data structure in:

- **categories** - each page or in other words article, can have unlimited children/parents.
- **comments** - the application supports nested comments to better structure the discussion.



## 6.6 Security

Security is an important factor in every single organization when it comes to storing their information. I kept this fact in mind and in line with the Ruby on Rails security features<sup>1</sup> and I took extra precautions that would help organizations keep their data secure.

These were: multitenancy architecture (previously mentioned), advanced authentication system (Devise), and user access with varying permissions.

### 6.6.1 Authentication Solution

Devise<sup>2</sup> is an authentication solution for Rails. It uses Bcrypt<sup>3</sup> to store a secure hash of your users' passwords. If a reader is interested in more details on cryptographic operations, Tiago Alves has published a detailed article on Medium.com worth reading<sup>4</sup>.

Devise is based on the modularity concept and this plays an important role in future development. Thanks to this characteristic it is fast to implement additional security features to better protect the data and users. Devise has multiple extensions<sup>5</sup>, some of the worth mentioning are:

- **devise\_aes\_encryptable**<sup>6</sup> - Adds AES encryption to Devise.
- **devise\_security**<sup>7</sup> - Adds options for industrial standard security demands.
- **devise\_two\_factor**<sup>8</sup> - Helps with implementation of 2FA authentication.

Every wiki can be accessed only after accepting the email notification. Here, Devise comes in handy again as it offers an extension<sup>9</sup> that helps with inviting new people and sending out the invitations. Another advantage of Devise's extensions is easy and fast implementation of signing up (or signing

- 
1. <https://rubyonrails.org/security/>
  2. <https://github.com/plataformatec/devise>
  3. <https://auth0.com/blog/hashing-in-action-understanding-bcrypt/>
  4. <https://medium.com/p/how-does-devise-keep-your-passwords-safe-d367f6e816eb>
  5. <https://github.com/plataformatec/devise/wiki/Extensions>
  6. [https://github.com/chicks/devise\\_aes\\_encryptable](https://github.com/chicks/devise_aes_encryptable)
  7. <https://github.com/devise-security/devise-security>
  8. <https://github.com/tinfoil/devise-two-factor>
  9. [https://github.com/scambra/devise\\_invitable](https://github.com/scambra/devise_invitable)

in) through third-party providers like Google. As of the beginning of the year 2017, there were more than 3 million businesses paying for Google's G Suite[26, 27]. These businesses can represent potential users of the application I have been working on. Hence, I can use Devise to implement authentication with Google and therefore attract more people to use the application.

Last, but not least, Devise also helps with handling edge cases like forgotten password, et cetera.

### 6.6.2 User Roles Management

Members of the organization usually carry different roles within this organization. I believe (and that is what my initial user research has shown) that this fact should be reflected in the application itself as well.

Therefore, I came up with five default roles a person can have within the application:

- **Owner** - a person who handles payments for the application, for instance, owner of the company. Has all the rights.
- **Administrator** - manages wiki itself, including content and members of the wiki.
- **Moderator** - can manage the content of the wiki. This is usually someone who already spent some time within the organization.
- **Contributor** - usually a newbie that can only create and edit own articles, join discussions, give reactions.
- **Reader (Guest)** - has only the permission to read the content and give reactions. This role is very restricted and suitable for newbies or people from different organizations (guests).

To better visualize permissions given by each role and to help with testing these permissions I have come up with this compact matrix:

ROLE	Reader	Contributor	Moderator	Administrator	Owner	
Can	Show the article					The red color here is WRONG.
	Show the user					
	Edit own profile					
		Create an article				
		Edit created article				
		Archive created article				
		Comment on articles				
		Delete own comments				
			Edit any article			
			Archive any article			
			Delete any comment			
				Invite people		
				Edit any user		
					Edit Wiki	
					Delete the wiki	
Cannot				Delete + edit the wiki		The green color here is WRONG.
			Invite people			
			Edit any user			
		Edit any article				
		Archive any article				
		Delete any comment				
	Create an article					
	Comment					

Figure 6.5: Visualization of permissions within the application.

These policies are put together in the 'access\_policy.rb' file which looks as follows:

Listing 6.3: A code example for the Reader role.

```

role :reader, proc { |user| user.present? } do
  can :read, Article
  can :read, User
  can [:read, :create], Wiki
  can [:edit, :update, :destroy], User do |
    edited_user, user|
    edited_user.id == user.id
  end
end
end

```

One of the challenges in implementation roles I came across was the fact that one person can carry on different roles in different wikis. This relationship is stored in *wiki\_users* table which carries 3 columns:

- **user\_id** - Identification of the user
- **wiki\_id** - Identification of the wiki
- **role** - a type of role

Because there are five roles to work with, the best way to interpret and later work with roles was to use attribute *enum* as follows:

Listing 6.4: Enum attribute describing roles.

```
enum role: { owner: 1, administrator: 2, moderator  
            : 3, contributor: 4, reader: 5 }
```

So that I can later ask about the person role as follows:

Listing 6.5: An example of asking if *current\_user* has role *owner*.

```
current_user.owner?
```

### 6.6.3 Password Strength Estimation

To ensure that new people joining the application will choose a password that is strong enough, I took advantage of Dropbox's *zxcvbn* library<sup>10</sup> to implement a live password strength estimation.

In the application it later looks like this:

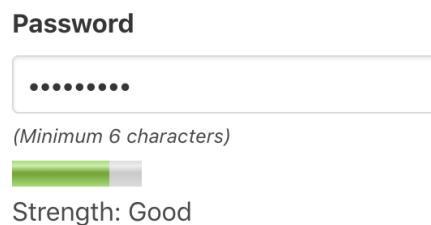


Figure 6.6: Live password strength estimation during registration.

There is the progress bar under the password field that updates with the input itself giving the person immediate feedback on how strong or weak his/her password is.

---

10. <https://blogs.dropbox.com/tech/2012/04/zxcvbn-realistic-password-strength-estimation/>

#### 6.6.4 Passwords from Data Breaches

After a person clicks on the "sign up" button during the registration process, the application validates the entered password against PwnedPasswords database.

PwnedPasswords project<sup>11</sup> collects all the released passwords from known data breaches allowing people to see if their password has appeared in any of those databases. That adds an extra layer of security and avoids common passwords.

### 6.7 Search and Search Analytics

As previously mentioned, to always ensure the most accurate search results, I decided to use Elasticsearch. To allow people to find the information they are looking for, I have extended the search not to only look through created pages but also through comments and users.

So if a person would look for the term 'Prague' it might result in:

- Page where 'Prague' is mentioned. For instance, a page about the Prague Office.
- Person somehow associated with Prague. For instance, a person who filled out the 'Location' field in his/her profile with 'Prague'.
- Someone's comment including word 'Prague'.

In order to perform analytics on the search, I would need to store information such as:

- **Query** - what a person typed into search input.
- **Results count** - how many results were returned on a searched query.
- **Created at** - when was the search performed.
- **Conversions** - I consider the event of clicking on the search result as a conversion which represents "finding the right result".

The results of the analytics allowed me to retrieve a set of the most searched terms with no results. While this might seem like a metric without value, it can actually provide great information to the person managing the wiki. For example, if members of the wiki were looking for a term "Design

---

11. <https://haveibeenpwned.com/>

Guidelines" way too many times and always ended up with no results, it might suggest that creating a page with this content could benefit the members of the wiki. This feature can hence serve as suggestions for topics to write about.

### 6.8 Testing Framework

Ruby on Rails usually comes with a testing framework called Minitest<sup>12</sup>. While Minitest provides a complete suite of testing facilities, I have decided to use RSpec in my work. RSpec is a *"Ruby domain-specific language for specifying the desired behavior of Ruby code."*[17] The main reason in choosing RSpec was the remarkable degree of code readability that it almost reads as a plain English text.

I can not think of a better way how to describe RSpec than show you an example code that tests creation of a wiki:

Listing 6.6: The code testing creation of a new wiki.

```
feature "User creates a new wiki" do
  scenario "with valid name and subdomain" do
    sign_in
    visit new_wiki_url
    fill_in 'wiki[name]', with: "Test_wiki"
    fill_in 'wiki[subdomain]', with: "test-wiki"
    click_button 'Create_Wiki'
    expect(page).to have_content('Wiki was
    successfully created.')
  end
  ...
end
```

---

12. <https://github.com/seattlerb/minitest>

## 7 Evaluation

Since my whole work was built around people and their needs, it could not end without the most important part — people. Therefore, once the first real prototype was ready (following design thinking phases) it was time to test it.

The goal was to reveal shortcomings that the prototype has, but to also discover opportunities for further improvements prior to launching the application.

Hence, I have met with five participants as a part of the user research to interview them in an informal and unstructured way about their opinions, insights, and suggestions for my work. To dive deeper into their minds, I asked a lot of follow-up questions and also used supportive questions that I had previously prepared.

Some of the responses were based on features that were not included in the prototype (see the scope definition section) because they were not crucial elements for the basic functionality of the application while some others made me think in a different way about the application and its future.

### 7.1 User Interviews

Overall, the interviews helped me validate that the requirements of an organization on my application differs based on the actual size of the organization. While smaller organizations mostly cared about attractiveness and new features, bigger companies did not even mention these elements and were more focused on important things like Single Sign-on, quick migration of the content from other solutions, et cetera.

Different types of growth strategies go hand in hand with the size of the company as well. I was told that to get my application installed at a new organization, a freemium model that would scale with the number of members of the wiki, would help. Other participants that worked for larger organizations, introduced the obstacle of a complex sales process for implementing new software.

This information allowed me to realize that I should pick the organizations I want to target with my solution first and lead the future development towards requirements of the targeted organizations to satisfy their needs to its fullest.

Another issue that stood out at the interviews to tackle is making the transition from one solution to another as smooth as possible since if the

organization does not start from zero, they probably already have some kind of solution for sharing information and collaborative work put in place. That means to research export options from most used solutions on the market and following the implementation of some kind of import functionality that would speed up the transition.

To sum the feedback from evaluation interviews I can say that people appreciated:

- The way the application works with data in wiki dashboards.
- Fulltext search that works across multiple entities.
- Social features like reactions, contributors profiles, and comments.
- Well separated rights based on the roles.

On the other hand, participants saw the main shortcomings of the application in:

- No ability for easy and fast migration from other solutions.
- Missing page templates that would either encourage best practices or follow the format they use within the organization.
- Missing Single Sign-on (only big organizations).
- Not that attractive and playful design.

Transcript of all the evaluation user interviews can be found in the Appendix B.

### 7.2 Software Testing

Since the performed user interviews helped me to better understand what people think of the application and helped me to set up the goals for the future, there is other important task when we talk the application and that is to make sure that the application is fully functional. To help me achieve this goal, software testing comes into play. This process excludes people's opinions and focuses clearly and only on the comparison of what we expect the application to do (for instance, create a new wiki) and what the application does when we try to perform this action (for instance, the application throws an error).



Because the application is mostly about basic manipulation with data (create, edit, update or delete), I found that the Unit Tests are not as applicable as writing Integration Tests. What I wrote are not just, so-called, Integration Tests, these are Feature Tests; tests interacting with the application just like a real person would do. When running these tests, system clicks on the buttons, links, fill in forms... to ensure that people get value out of the application. Or in other words, these tests are making sure that the application is not broken in such a way that a user might notice.[28] *"Because feature tests need to mimic the user's behavior as much and as close as possible they are using a web browser to do their work and that is very slow, which is their biggest downside."*[28] On top of that, the application was tested by manually performing various actions step-by-step closely watching whether the behavior of the application matches the expected behavior.

### 7.2.1 Feature Tests

When writing feature tests I have decided to cover *the most crucial parts* of the application allowing people to perform the most important tasks. These included:

- Signing up and signing in.
- Creating a new wiki.
- Creating a new page.
- Joining a discussion by leaving a comment.
- Updating a person's profile.
- Reacting to a page.

A test case for creating a wiki looks as follows.

**Test Case ID:** 03\_user\_creates\_wiki\_spec. Based on the use case MH001.

**Test Case Description:** A user creates a new wiki.

**Created By:** Marek

**Version:** 1.0

**Prerequisites:** A user is signed in.

**Test data:**

- user[email] = valid@exampler.com

- user[password] = hlavneheslo
- wiki[name] = Test wiki
- wiki[subdomain] = test-wiki

**Test scenario:** User creates a wiki with a valid name and subdomain.

**Expected result:** The wiki will be created. This will be confirmed with the notification including the text: "Wiki was successfully created."

**Steps:**

- Navigate to `http://lvh.me:3000/u/sign_in`
- Enter user[email] and user[password]
- Navigate to `http://lvh.me:3000/wikis/new`
- Enter wiki[name] and wiki[subdomain]
- Click on the "Create Wiki" button
- Expect the final page to show the notification with the text "Wiki was successfully created."

A simple example of the feature test with different scenarios testing the edge cases is provided below.

Listing 7.1: User creates a wiki feature test.

```
feature "User creates a new wiki" do
  scenario "with valid name and subdomain" do
    sign_in
    visit new_wiki_url
    fill_in 'wiki[name]', with: "Test_wiki"
    fill_in 'wiki[subdomain]', with: "test-wiki"
    click_button 'Create_Wiki'
    expect(page).to have_content('Wiki was
      successfully created.')
  end
  scenario "with valid name and invalid subdomain"
    do
      # Should pass as the application is able to
      # sanitize the subdomain string before saving
```

```
sign_in
visit new_wiki_url
fill_in 'wiki[name]', with: "Test_wiki"
fill_in 'wiki[subdomain]', with: "test_is_good"
!
click_button 'Create_Wiki'
expect(page).to have_content('Wiki_was_
    successfully_created.')
end
scenario "with_blank_fields" do
  sign_in
  visit new_wiki_url
  fill_in 'wiki[name]', with: ""
  fill_in 'wiki[subdomain]', with: ""
  click_button 'Create_Wiki'
  expect(page).to have_content("be_blank")
end
end
```

After running the tests against a testing environment, all the tests passed what proves that *tested functionality meets the requirements*.



## 8 Deployment

The way to deploy the Ruby on Rails application to the internet is very easy and fast with the platform called Heroku<sup>1</sup>. It simplifies the whole process by providing numerous tools like command line tools<sup>2</sup>, linking with Git repositories<sup>3</sup>, and rich documentation<sup>4</sup> on deploying web applications written in various web frameworks and programming languages.

When I was about to use Elasticsearch as a supporting pillar for full-text search I was not sure whether this decision will not introduce future issues with deployment. Luckily, with Heroku, this is only a matter of two clicks since Heroku offers plenty of extensions one can use to introduce additional features to the servers. One of these extensions is also an extension<sup>5</sup> that gets Elasticsearch instance up and running.

If there would be a need to deploy the application to a different platform the process is no different from deploying a regular Ruby on Rails application and getting the Elasticsearch server up and running. Many of the technology stack decisions are highly abstracted in configuration files in Ruby on Rails so it is easy to combine multiple providers of different services together. For instance, it is possible to have one database provider, another place where the actual application would run, and another place where the application would store uploaded data.

- 
1. <https://www.heroku.com/>
  2. <https://devcenter.heroku.com/categories/command-line>
  3. <https://devcenter.heroku.com/articles/git>
  4. <https://devcenter.heroku.com/categories/language-support>
  5. Bonsai Elasticsearch — <https://elements.heroku.com/addons/bonsai>



## 9 Future Development

The application is currently under active development. The next steps are directed towards implementation of more features; listed in the scope definition section; and making the application available to the first real people to try it.

While the design was one of my capstones to innovate in, and I did my best to make the design intuitive and usable, I am not fully satisfied with the architecture that powers front end. Hence, after proving a product market fit of the application and getting some initial traction I would love to separate front end from back end creating a reactive user interface <sup>1</sup>. The bright side to the current user interface is the fact that Ruby on Rails uses so called turbolinks. Turbolinks is *"JavaScript library that, when enabled, attaches a click handler to all links of an HTML page. When a link is clicked, Turbolinks will execute an Ajax request, and replace the contents of the current page with the response's <body> tag."*[17] Turbolinks makes the application appear faster and reactive even though there is no well-known user interface framework like React, Angular, or Vue in the background. Another advantage of Ruby on Rails is that it serves the format you request. Hence, it does not require any extra line of code to serve data in JSON when requested, allowing for future REST API approach.

Lastly, after hearing out opinions of people who saw the first prototype of the application and shaping a better picture of the course, I should follow, I would love to catch up with automated tests as the application is getting more and more complex and even in the last days before submitting this work I felt that it is harder and harder to keep an eye on the correctness of every tiny feature of the app.

---

1. <https://medium.com/modern-user-interfaces/a-journey-into-reactive-user-interfaces-101-1daea5702486>





## Summary

Based on the the initial user research, competitive analysis, and conducted user interviews, I, as well as participants of the research agreed that *there is room for improvement* in wiki software space.

In my user research I have learned from participants that the most important things to them are *discoverability, searchability, and rights management*. The problems that people using Wiki software face are that the *information gets obsolete fast*, and there is rarely a dedicated person who would keep the wiki up to date.

Following Design Thinking phases, I empathized with people using Wiki software by the *conducted user research* (Empathize phase of Design Thinking), I have *defined the problems, generated ideas* how to solve them (Define and ideate phase of Design Thinking), and brought those to the *low-fidelity wireframes* which I right after *turned into the code* (Prototype phase of Design Thinking) using latest technologies found on the market.

Once the prototype (the application) was ready for presentation, I have conducted *evaluation user interviews* with five participants to validate (Test phase of Design Thinking) with them that my prototype *fulfills the requirements* for a modern wiki web application.

The participants pointed out things they enjoy in the application, and helped me to gather feedback and insights for the future development of the application which I plan to actively continue with to *present the application* to the first, real customers.

To me, this was an excellent opportunity to get my hands on whole *product development cycle* from definition, through scoping, all the way to the development. I learned how to *develop web applications* fast using the modern Ruby on Rails framework, and how to *resolve issues* connected to *shipping a real world product*.



## **A Initial User Interviews**

**Goal:** Get to know what people use inside of their organizations nowadays to share, gather and access knowledge. What they like about those solutions and what they don't like. What is missing and how the perfect software for gathering and sharing information should look like.

### **A.1 Supporting Questions**

#### **A.1.1 About Participant**

- What size is the organization that you currently work for? And how the structure looks like?
- How long is the mentioned organization around?
- What does your typical weekday look like?
- What are some of the apps and websites you use on a regular basis?
- Tell me about your role at your company?

#### **A.1.2 Product Related Questions**

- In the terms of information gathering and sharing across your company. What are the things you are trying to get done? Why? (Ask why few times in the row)
- How often do you seek an information regarding to your work/existence in the office? What information you seek?
- Where and how do you access these information? What tools you use?
- What other products or tools have you tried out?
- How did you hear about these other products or tools?
- What do you like or dislike about these other products or tools?
- How much time do you typically spend on finding a right information?
- Tell me about the last time you tried to find some information?
- What do you like about how you currently search for the information?

## A. INITIAL USER INTERVIEWS

---

- What's the biggest pain point related to finding the information?
- What are you currently doing to make this task easier?
- Have you ever shared information helpful for more people within your organization? What was it?
- What do you like about how you currently search for the information?
- What's the biggest pain point related to sharing the information?
- What are you currently doing to make this task easier?
- What is the hardest part about having information together at one place?
- Are you looking for a solution or alternative for finding and gathering the information?

### A.1.3 After showing 2 Wireframes

- What do you think of this product?
- Why do you think someone would use this product?
- Now imagine this is a page where you would land. It's your new company knowledge base.
- Is there anything you are missing on this screen?
- What could be done to improve the experience?
- Is there any information you would be missing on this screen?

## A.2 Interviews Transcript

### A.2.1 Participant no. 1

A Startup ( $\pm$  50 people) recently acquired by big corporate for \$ 100M.

A middle level management person, responsible for information management among many other things.

We distinguish 2 types of content in your wiki:

- True, long lasting knowledge that will be up to date even after 5 years

- Temporary information (current projects, company news — we use GDrive for it)

The big question is **how to manage the Wiki**:

- How to deal with moderators? Who can maintain the content?
- How to motivate people to contribute?
- How to motivate people to contribute with quality content.

(I showed him the roles proposal we have had.)

Reaction: If some people won't be able to contribute → system will die.

Super important elements to him:

- Ability to search, good search (fast, full text)
- Discoverability (tree structure extremely important)
- Authentication (definitely SSO and/or Google Auth + public links when you want to share the article with someone out of your organization)

Said that versioning is also very important to him, but after more deep discussion he accepted that he doesn't use it that much, but it would be great if it will be available (just in a case). Plus, showed me the Dropbox Paper History Changes page which was done very well.

What they use?

- Dropbox Paper — Why?
  - He likes the document navigation on left side.
  - He likes versioning view
  - Loves WYSIWYG editor — markdown support, todo lists, code snippets, mentions
- In the past he has an experience with Confluence
  - The only thing he liked about it was creating the boxes which help to highlight certain information (imagine those as simple rectangle with colorful background)

He likes the Mac OS X spotlight search a lot. The same in Evernote and Slack (a quick access to basically everything).

## A. INITIAL USER INTERVIEWS

---

He expressed how much he likes shadow under the images. I asked him about image captions — said it's nice to have but he doesn't use it often. Seems that aesthetics is important to him as well as user experience.

The problem he sees with Dropbox Paper is that in the case he wants to share a folder he had to first share every single document. Would be great to have an option to share any item in the tree (folder or document or file. . . )

I asked him about having multiple “spaces” (wikis) within one Wiki. (It's how Confluence is done). Said, that it made sense in his previous company (a bank) with over 130k people and it might fit also to a company that acquired them recently with 1600 people. But not for them — a small startup. His argumentation was that at some point you will end up in the stage where what others within your company do, it's not important to what you do. He said, after that it would interesting to have distinguished homepage that would separate him from others.

I showed him couple in process wireframes:

(Article)

- Breadcrumbs missing
- People scan, don't read (10 min to read is unnecessary)
- Discussion at the end of the article and in-context comments feels duplicating the same “feature”
- “It remains me a blogpost a bit” (because of related articles at the end)

(Homepage)

- Loved articles pinning (from time to time there are articles you access twice a day)
- Recent activity (would be great if it can be tailored to his needs)

### A.2.2 Participant no. 2

Works on marketing within small startup ( $\pm$  30 people) that has been part of prestigious 500 Startups accelerator.

They have used Gitlab Wiki for their projects and teams before. Now they mix Dropbox Paper and Gitlab Wiki.

Very important is to **keep information up to date** and **manage who has access** to what information.

He feels like Dropbox Paper is more advanced and better than G Docs. He loved the feature in Dropbox Paper that user can actually open the image and comment on it (where he clicks he creates a new comment). Dropbox Paper supports markdown, mentions (not only in comments but everywhere), tasks. . . Dropbox Paper also remember the authors of every line, even when user is copying or moving something across 2 different documents. He really likes versioning. It's more human, not like in typical solutions where people have "Tuesday 2.10." and some message.

Everybody on their teams work based on issues assigned to them. They link issues to the Dropbox Paper as well. He thinks that every strict structure for small startup is bad. Strong need for collaboration, changing things.

He used mind mapping a lot but it was not enough for storing all the information.

He said that lot of companies already has some starting point regarding to their knowledge therefore the migration is really important.

Also expressed the importance of templates that help people to quickly create new content without need to think of the format. (Template examples: Meeting Notes, Brainstorming, Checklist, etc.) It is easier to start with something pre-made than with blank page.

3 examples of knowledge (ideas for templates):

1. Values, high level knowledge, company knowledge base
  - (a) e.g. working hours, Wi-Fi password, printers setup, tools company uses — document that you would send people on their first day
2. Team level knowledge
  - (a) specify toolkit, regular meetings (who owns them — very important)
3. Knowledge for people that spend some time with company — checklists for deployment, writing e-mail campaigns, etc.

"Those three templates would made creating knowledge base easier"

At the company he works for processes distilled from the bottom of the company. Half year back they have started working in the sprints. Year back they have started using OKRs. They are collecting a lot of data but doesn't have anyone who would evaluate them.

Interesting idea for him is to write about best practices for young firms to create their knowledge base (on the blog as a content marketing). Tips what

## A. INITIAL USER INTERVIEWS

---

are the sprints, what are the OKRs. How-tos. Interesting market of small startups and firms that are just about to set their company processes.

The important element for him is also integration with other services (G Calendar, Gitlab, etc.). Last interesting idea was that people might ask for permission other users and those can grant the access.

After showing him wireframes:

(Article)

Code highlighting is crucial

Component for article feedback could be fixed since user doesn't go to the end of the article always

(Homepage)

Recent activity, super important

Didn't get the meaning of the tags under the search box

Recent activity — might feature the select box with team-wide, company-wide options so that he can see only the relevant stuff.

Files in the tree structure should feature timestamps so it's clear when they were lastly updated.

### A.2.3 Participant no. 3

10 years in huge corporate with 84k employees. Been writing a strategy about knowledge management for the CEO in 2007. Now founded his own startup.

Back in the days people used e-mail to share the knowledge and it's widely used also nowadays. About the year 2005-2007 wiki forums become a word. They have reduced the cost of support for example. People could answer their questions themselves. Then the same principle was translated internally to the companies. But everyone within the company is focused on their own work. So the motivation for people to contribute with the content is not that strong. Usually the ratio of contributors vs. people viewing the content is 4% : 96% .

In his startup they use wiki as well but the informations got **out-dated quickly**. People move to another project/task and the ownership is not transferred. The system is passive, doesn't drive anybody to update the content or archive it.

Google Doc is great for collaboration however it's terrible for keeping things up-to-date. This problem simply hasn't solved yet.

Slack and other tools have come but they don't solve this problem, it's hard to search for the knowledge. He sees the future in NLP (natural language processing) and machine learning — so that software would figure



out itself what is the knowledge. Would be great if we can create a knowledge from unstructured conversations (Slack).

Has pointed out to the guy from NYC writing for Wall Street Journal that they have a models that predicts what authors should write about and what people are interested in. Might be our case — to be able to **suggest people what to write about**.

#### A.2.4 Participant no. 4

A software engineer working for the most famous internet search engine in the world.

They used to use their own internal wiki (custom made). But now they use only markdown files that lives within software projects (mainly for documentation) — called g3doc that will generate documentation from those markdown files. You can also link that markdown with the code and all documentation have the same look. The system enables to generate docs right away from the code, almost like JavaDoc.

Across the company they have this “super search”, an internal tool that can search across the whole huge code base and everything within the company. (again we can see the repeating pattern and the **importance of search**)

For the information not regarding to the code they have programmed their own robust intranet. He said it’s also possible to navigate through the “golinks”, where user would type in the browser go/eat and it will immediately show the menu for lunch for that day. Golinks are basically shortcuts for different web sites.

There is four ways how to store information:

- Code (g3doc)
- Things like: “How to make a coffee” — using their own solution for creating websites (also pretty famous in the public) but trying to make transition to g3doc
- Another option to “How to make a coffee” information might be their own Docs system. They use it also for solution design.
- Every single project has their own web site and dashboards.

He said, he uses search almost always. Don’t really care about outdated information but strive for having the best search out there so they always find the right information across the whole company. The problem with internal search is that ranking pages is harder since they doesn’t have enough data.

Participant typically search in the code. Last time he searched for something, he was looking for the presentation he attended — internal search and then their own Docs engine where he found the presentation. He wrote few documents to propose better solutions within company (using their own Docs tool), sent it to the team mailing list and allowed sharing (which means that also the internal search indexed the document).

### A.2.5 Participant no. 5

CTO of an smaller size startup ( $\pm 50$  people) recently acquired by a big corporate.

He thinks that the problem with knowledge bases was still not solved.

He saw one solution in the past that he really liked — Corilla<sup>1</sup>.

There is a big difference if somebody from a company is specially dedicated to maintain the system or not. Hardly depends on the size of the company. (Why?) Because when there is somebody taking care of the docs and knowledge base the steeper learning curve doesn't matter that much.

(I feel like he was shifting the whole interview to the totally technical side and to writing the docs for code. But we would love discover this area as well.)

Pointed out to readthedocs site<sup>2</sup>. They use sphynx<sup>3</sup> — docs engine for python. Absolutely technical, only for developers, tech audience.

He distinguish 2 types of content:

- Unstructured — who is on my team
- Content related to code

For him the ideal solution would be something that can be well connected with the code.

All the big players use Confluence including the corporate that acquired them because it works good with JIRA. However, the corporate has **dedicated team that maintain that knowledge base**. That is the only case when Confluence works well.

He saw a solution where people would write in the format heading + paragraph so that it can be reused on the other places.

He said that bigger firms require **on-premise solutions** because of the regulations and etc.. **Docker** has helped this a lot. There is also one startup that can help develop on-premise solutions.

---

1. <https://corilla.com/>

2. <https://readthedocs.org/>

3. <http://www.sphinx-doc.org/en/stable/>

### Spynx

- Pros:
  - integrated with code
  - it's modular
  - their own structured language (enable links, embedding so that the same information change across the whole knowledge base)
- They also do code reviews for documentations (someone would check if what you have written to the docs is correct) — The con is work load but pro is that undesirable content won't show up in the docs.
- Cons:
  - syntax of the language is not that good (markdown would be better)
  - the **design of skins and layouts can be significantly improved** (information architecture is poor — navigation, search)
  - big pain to work with images (first he has to create image on the side, push it to the repository and then link)

### Confluence

- Cons:
  - Information architecture is poor (a lot of unstructured information everywhere)
  - Doesn't guide user to write structured and relevant
  - Missing feedback loop

In their startup there was never a person that would be dedicated to maintain the knowledge base.

Since we were talking more about code docs, when code has changed the docs changed with the code. So the docs remained up-to-date.

Showed him few wireframes (Article)

- Don't put the discussion under the article. Do it as a "tickets" /issues that need to be resolved "in the article" . User doesn't want to get back to the discussion.

(Homepage)

- **Recent activity is super important.** Should be relevant, not spammy.
- **Search is super important.** He doesn't care that much about tree structure because he believes that a person is not going to read everything but look for one exact information (however, we already know from the previous research that tree structure might be important if a person writes a new article and doesn't know into which category it falls.)

### A.2.6 Participant no. 6

Software engineer at the most famous social network nowadays. Working on the social network for organizations to help them get the work done.

He said, in the past they used to use their social network for everything (groups and other features). It worked well so they have pivoted social network for work. Personal social network account is totally separated from the work account (people like they separate their own privacy from work).

Not everything is stored on this social network. They also have wiki pages (mainly for "manuals"). **Very good internal search** (can search through the social network groups as well as through the wiki pages). They use more systems but because of the great search it doesn't matter where the information is stored.

When developing new product / working on the new project they create 3 groups:

- Information (FYI, announcements)
- Feedback Group (for providing a feedback)
- Internal (closed for people working on the project)

They also have Q & A groups (e.g. iOS) so that everyone who is interested in this technology/architecture/product can ask and relevant people will answer. The internal search index those as well. He likes that when starting to write a new question, **system will first try to provide him similar questions** that has been already asked and are already answered. This preserves duplicates and speed up the process of finding the information.

They do have 2 wiki engines. One is modified open source solution and another in-house made. In the left sidebar there is navigation through chapters. It supports markdown. User can generate document outline.

His using search almost all the time. Last use case: been coding something, needed to modify the part of the code base so he tried to search for

some keywords, asked in the group and they have linked him to their internal wiki.

He sees writing a wiki pages as a time saver. So next time when somebody writes him a message he can just simply link that person to the page he has created.

He, as a person who consume the information most of the time, often edit the pages. Actually, **it is surprising if something is not out dated**. To update the page it's just simple click, edit, and submit. No need to review, approve content. People are happy that someone has edited it.

(Asked if it isn't problem that the pages are out dated) Depends on the pages and the frequency of the usage. More used the page, bigger problem it is and vice versa. When the page is out dated it means more work for a person because he/she has to explain the up to date information.

They use source control Mercurial, Q& A wiki, FAQs. . . He usually came across dev manuals.

(What kind of information do you usually store?)

- static information
- dev manuals
- information about offices, basic information (expense policy)
- printers
- organizing events, company sponsoring, criteria, rules. . .

No templates provided in their wiki solution. Meeting notes are usually posted to the group, so that people gets notified and can comment on it.

Using Quip — something like Google Docs. Good for real time collaboration in one document/table. Use case — planning or a draft for an announcement.

(Problems you came across when looking for the information?)

- hard to figure out if any wiki page exist for a given search term
- hard to say what to search for — e.g. looking for a answers on “how the xy system works” — then he would go to the group and ask there.

Wiki is wild environment, no one pushes you to contribute, but you want because it saves a time. When the content is out dated you can contact the author and ask him. You can see who have contributed in the past, the whole history of contributors. He has contributed in the past and then people asked him for advice.

## A. INITIAL USER INTERVIEWS

---

Usually the media types are code snippets (dev manuals), images, nothing special.

## **B Evaluation User Interviews**

Goal: To find out what went well and what did not on the first prototype of the application. See whether important features from initial user research were implemented sufficiently, and document what other things might block potential customers from using the application for their activities.

### **B.1 Supporting Questions**

#### **B.1.1 Product Related Questions**

- Why do you think someone would use this product?
- What might keep you from using this product?
- What is the most you would be willing to pay for this product?
- Does it remind you of any other products? If so, what products?
- What is most appealing about this product?
- What is the hardest part about using this product?
- What could be done to improve this product?
- Was there anything missing from this product that you expected?
- What do you think of this product?

### **B.2 Interviews Transcript**

#### **B.2.1 Participant no. 1**

The participant is an indie hacker who loves to create and run projects with his friends.

He thinks that the product could be useful to store the information that should be accessible by all the members of his team. Would consider using it for his brand new idea.

As someone who has just an idea, he obviously does not have much money for new tools so for him to make it attractive the solution would have to be for free, at least at the beginning. He suggested at usually when he is about to start something new, there is only a little portion of the people working on the project, therefore, it would be able to build the pricing

model around a number of members of the wiki. New startups should definitely get an advantage over companies that already make some revenue. Lastly mentioned that freemium business model always worked the best in his projects.

I asked about other products that it might remind him of and he pointed out Atlassian's Confluence and mentioned he heard about some open source solutions but is not sure in what stage they currently are.

Going through the application, wiki dashboards really stood out to him, as he liked the idea of working with the data. Especially, he appreciated the suggestions to keep the wiki up-to-date, review old pages or poorly rated pages. After trying out the search he loved an idea of searching truly all the content on the wiki — users, comments and pages.

He could not really point out any serious usability issues that would restrict him from using the product. Although, he mentioned that as a brand new organization he would love to have some kind of template showing the best practices or pre-made structure to keep the information. In the end, he mentioned that he sees a room for improvement in design and especially interaction (for instance, adding animations et cetera).

### B.2.2 Participant no. 2

The participant works at a large company with thousands of employees.

"I can definitely see the use of this solution in any kind of organization. From the company that I work for which employs thousands of people to nonprofits to make their lives easier."

He dived deep into the process of big enterprise companies, their requirements and how do they choose the solutions. Briefly:

- To get such a product to the company, I would need a serious sales team that would be able to negotiate with decision makers of the customer company the terms and conditions. This process can usually take months and it is hard to find the right decision maker in the company.
- Really important are also contracts and legal documents. Without real support from lawyers, proving the company that you really can protect the data they would be saving to the wiki you won't succeed.
- Many big companies are starting moving to cloud which is good for your application, however, not all of them. You might consider providing an option to install your solution on own servers.



Did not want to discuss the price for such solutions in big companies.

When showcasing the application he noticed a couple of things that would stop big companies from using this product:

- Missing Single Sign-on.
- No possibility for sending out the mass invitations to the desired members of the wiki. Mentioned also a possibility for restricting the registration to only people within the same domain (e.g. @acme.com)
- Very critical and yet not resolved issue he sees is that any larger organization already has some content put in the place. However, he does not see any option for the new administrator to import the content to the wiki. According to him, it is crucial to make the transition from one solution to another one as easy as possible.
- Missing feature flags. While this might not be a deal breaker he still thinks it is important for larger companies to have control over what features are enabled and which are not.
- Lastly, he liked the idea of wiki dashboards and dictionary of contributors, though, he said that usually larger organizations have dedicated solution to handle this area.

### **B.2.3 Participant no. 3**

The participant works at a startup with up to 50 employees.

The one crucial part he sees missing is the import of the content. As their startup already uses Confluence it might be too time-consuming to copy and paste all of their content into a new solution. That's what would hold him back from using my solution.

Since there is plenty of open source solutions he would definitely not pay if he would be starting out a new company. However, said that in a later stage, he would be willing to pay up to 30 dollars per month. Suggested to base pricing on the number of members of the wiki.

He has already heard about people using Dropbox Paper as a place to store the information accessible by anyone within the company but from traditional solutions he mentioned:

- DokuWiki
- MediaWiki (Wikipedia runs on MediaWiki)

- XWiki
- Confluence

The most appealing thing to him is the way the application works with data and encourage users to take the right actions, leading to a healthy ecosystem. Also liked the way people are able to react to pages, however, mentioned that there might be no need to react to some of the articles with long-lasting knowledge. Reactions might be more interesting for pages that are relevant at a certain point of time (note: touches on feature flags).

Expressed that he misses the ability to “watch” the page or in other words be notified when there are changes to the page. Would love to see some notifications system within the app as well as in his email inbox.

### **B.2.4 Participant no. 4**

The participant works as a designer in a company with up to 20 employees.

“The visual of the application look way too basic to me with no personality.” He would welcome a “cooler” design like Apple native apps have — more animations and interactions. Making it more personal. Even suggested using a serif font for pages to make them easier to read and to distinguish the look from other solutions.

While writing creating new page he suggested an ability to create tasks, in other words, a list of checkboxes so that he could use the application to manage his work. Another block that would come handy to him was the ability to create a table (this are currently not supported by the WYSIWYG editor).

He liked the possibilities of the full text search as he pointed out that this future is crucial.

At the end mentioned he misses an ability to create own pages templates which would follow the format of the documents they use within the company like meeting notes, checklists (for instance, before releasing a new project). When saw the comments at the bottom of the page, he expressed the opinion that these are usually useless as he would like to comment in-context like in Google Docs when, for instance, there is some mistake in the page that needs to be fixed.

### **B.2.5 Participant no. 5**

The participant works at a large software company with hundreds of employees.

One thing that would hold him back from using it in the company he works for is the missing Single Sign-on, as all the other software that they use, support it and it is not acceptable for them to be creating brand new accounts.

I received a good feedback on very precisely divided roles and permissions linked to these roles. However, the participant also mentioned that in the large organizations, sometimes you do not want one team to see something that other team can see. Therefore, he would work around this idea in the future if the application would aim for enterprise space. He talked about having special privacy settings for each page + introducing concepts of the user groups that would have assigned roles.

Lastly, when browsing pages he mentioned a problem / an opportunity as his use case is that, often, when storing the knowledge, not all the related information are text files. According to him, it would come handy if one would be able to attach or embed files to pages too, easily preview them and download them or upload revision at any point of time.



## C Wireframes

### C.1 Landing Page

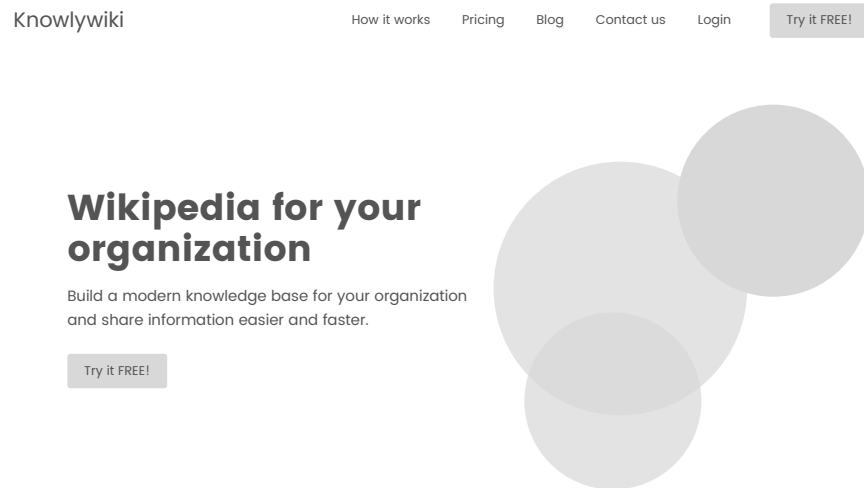


Figure C.1: An example of the future landing page.

## C.2 Onboarding

### Create your account

Register with your work Google account

G Register with Google →

Full name

E.g. Marek Dlugos

Work email

E.g. marek@acme.com

Password

At least 8 characters

Register →

By clicking "Register" you agree to Knowlywiki's Terms of Service and Privacy Policy.

Photo of a person

"Helpful testimonial from the current customer."

Figure C.2: Creating an account.

## New knowledge base

Name of the knowledge base		Usually the name of your company.
<input type="text" value="E.g. Microsoft"/>		
Claim your site		Must be lowercase characters. No spaces. Can't be changed after.
<input type="text" value="E.g. microsoft"/>	<input type="text" value="knowlywiki.com"/>	

Note: Use JS to fill  
in the domain  
name based on  
the name of the KB

Figure C.3: Creating a first wiki.

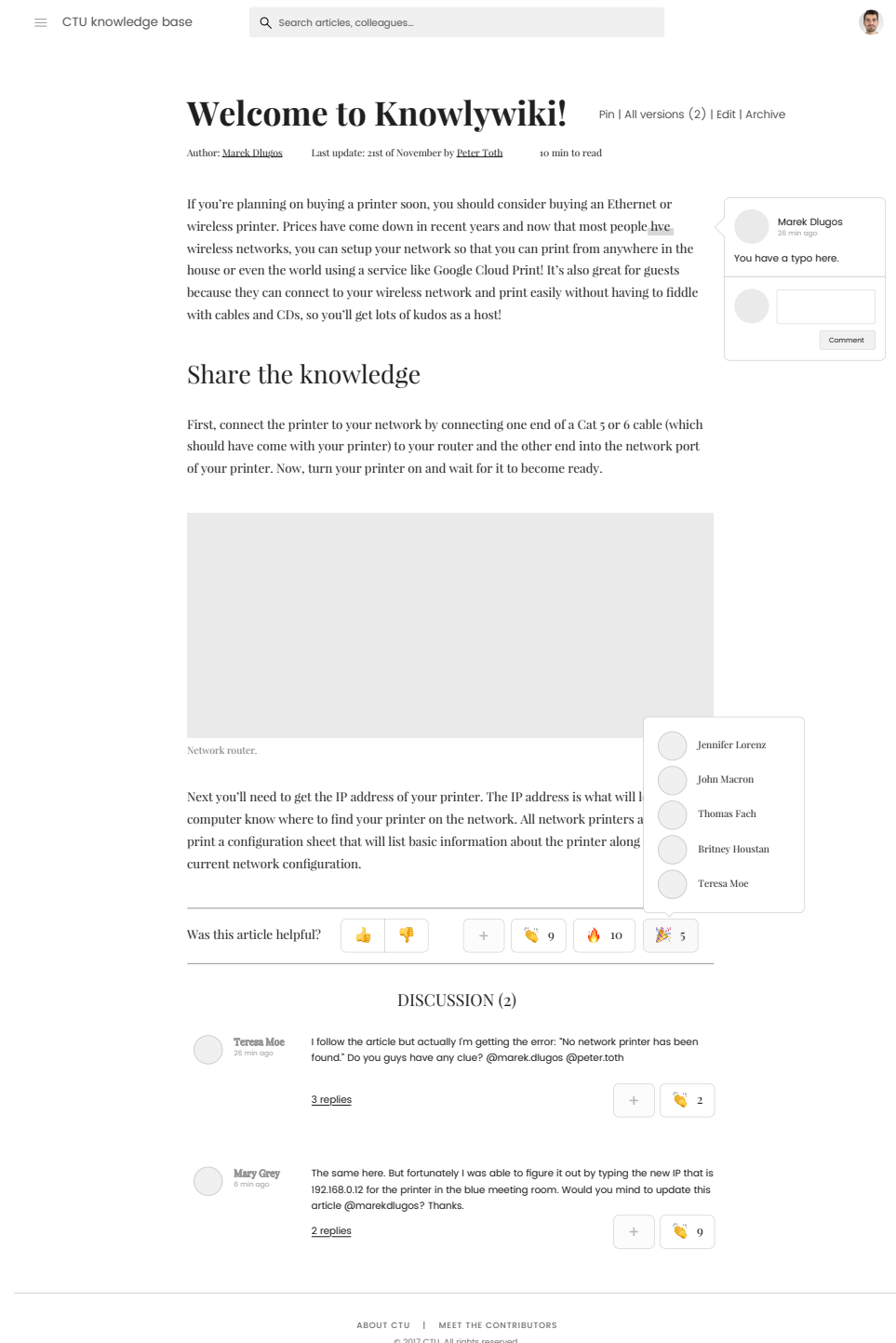


Figure C.4: Welcome screen showcasing application features.



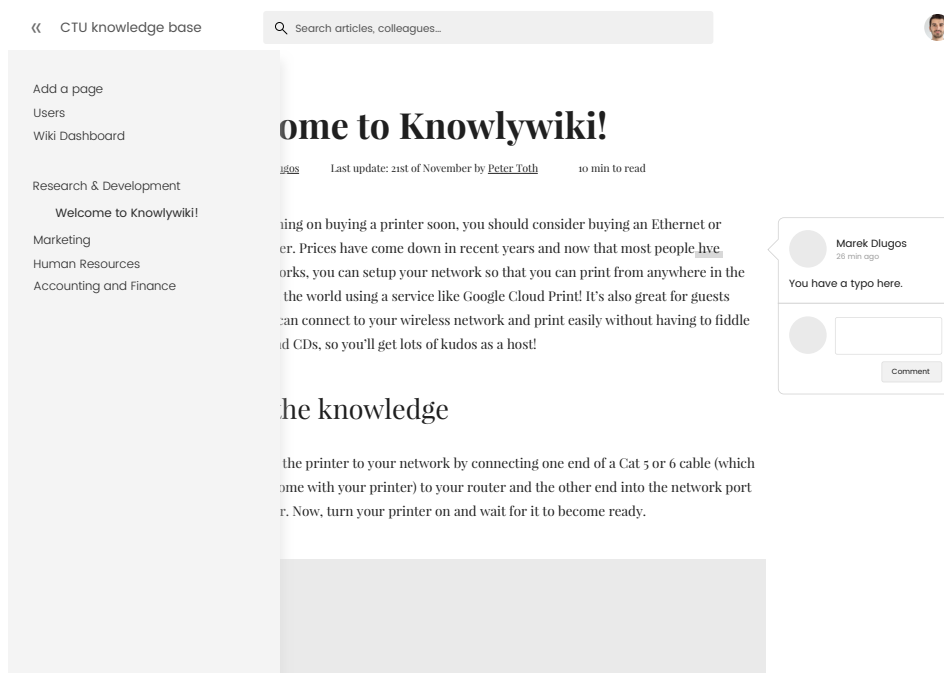


Figure C.5: Example of the menu rolling out from the left.

### C.3 Wiki

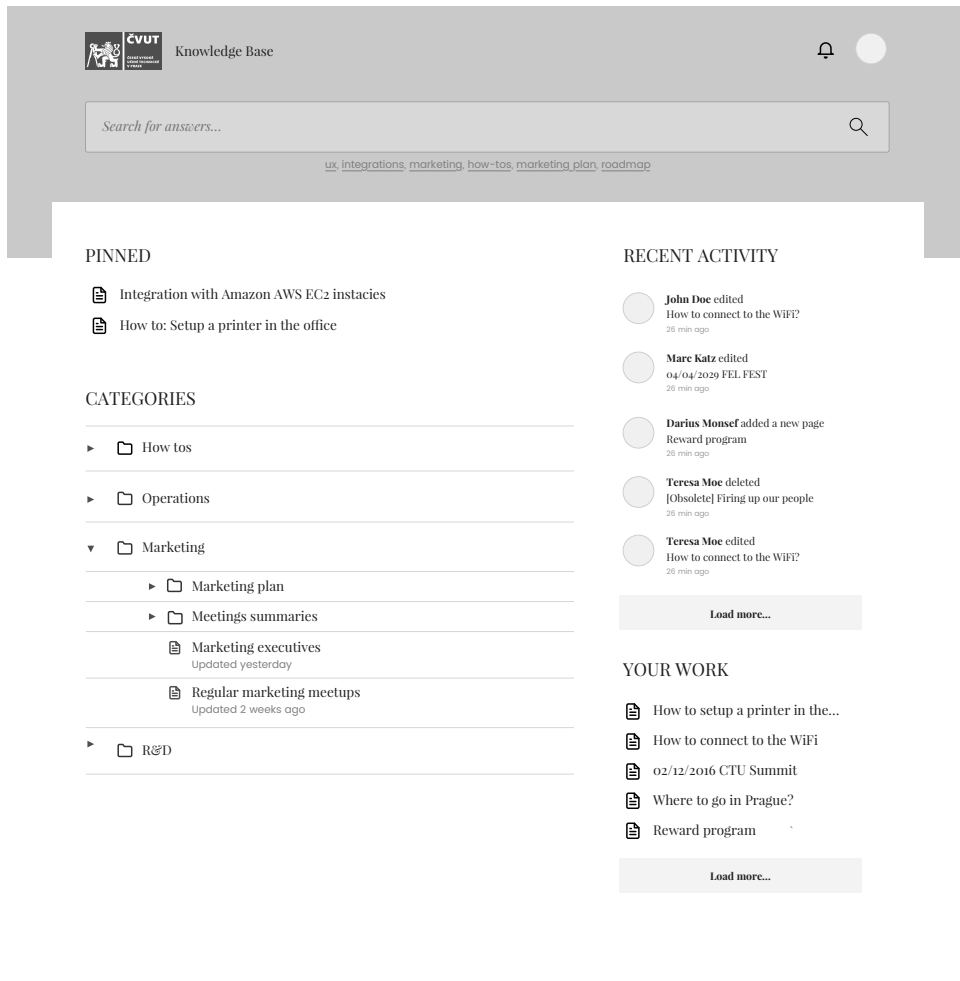


Figure C.6: Example of an homepage after login.



Figure C.7: Example of an article.

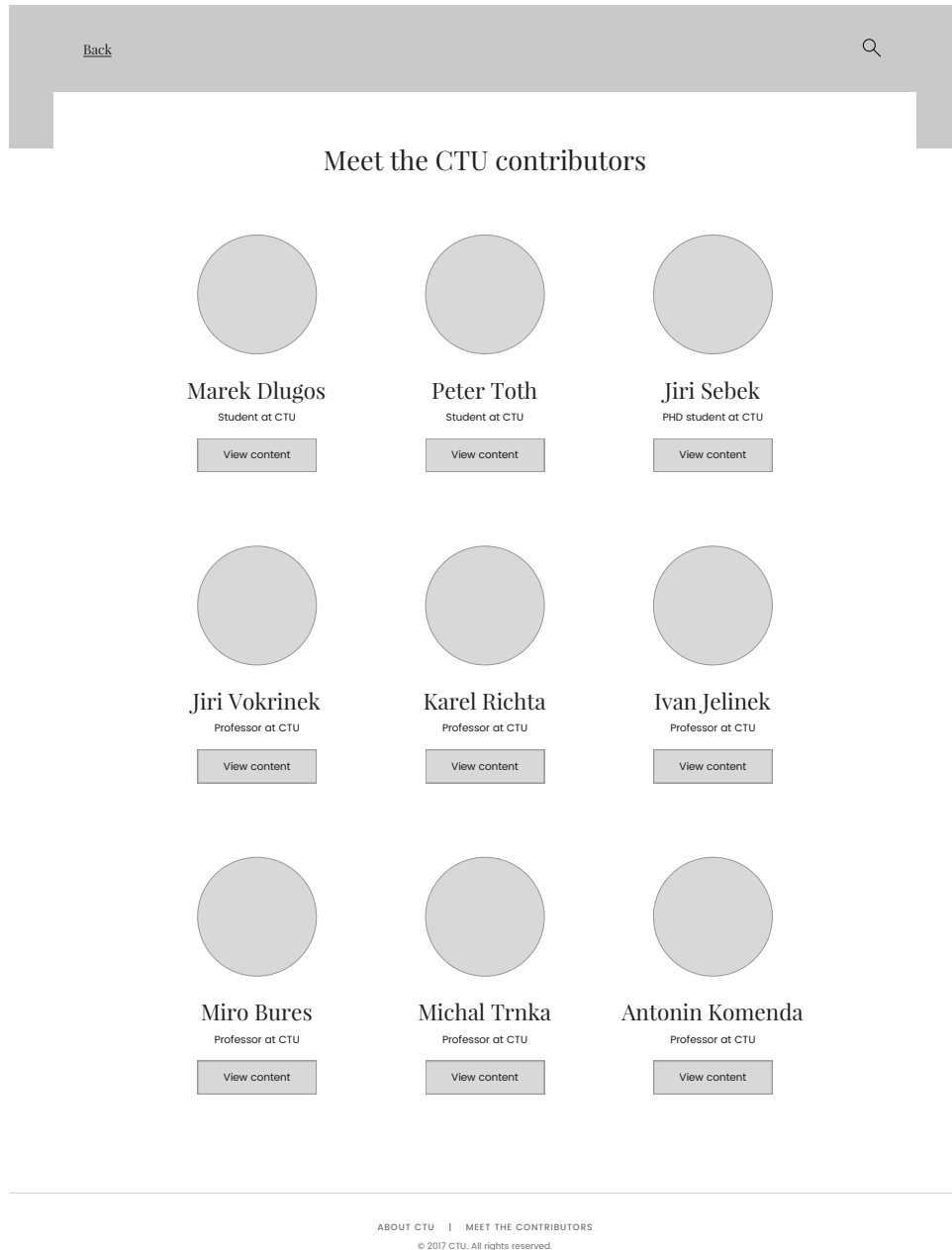



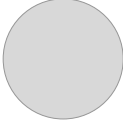


Figure C.8: List of contributors to the wiki.

 Knowledge Base





**Marek Dlugos**  
Student at CTU

Edit profile

Designer 21yo • Living in Prague 🇨🇪 • Originally from 🇸🇰 🇨🇪 Amateur self-taught photographer 🌍 Traveling the world 🧑🌍 See the world' beautiful destinations






### Organisation

Position	Student
Dept	Software Engineering & Technology
Location	Dejvice, Prague

### Personal


E-mail	<a href="mailto:marek.dlugos@fel.cvut.cz">marek.dlugos@fel.cvut.cz</a>
Website	<a href="http://www.marekdlugos.com">www.marekdlugos.com</a>
Tel	+420 895 983 233

### Activity

-  How to setup a printer in the office  
Edited yesterday at 4:32pm
-  How to setup a printer in the office  
Edited yesterday at 4:32pm
-  How to setup a printer in the office  
Edited yesterday at 4:32pm
-  How to setup a printer in the office  
Edited yesterday at 4:32pm
-  How to setup a printer in the office  
Edited yesterday at 4:32pm

ABOUT CTU | MEET THE CONTRIBUTORS  
© 2017 CTU. All rights reserved.

Figure C.9: Example of a contributor profile.

 Knowledge Base

Sign in

E-mail address

Password

Forgot password?


Login

---


ABOUT CTU | MEET THE CONTRIBUTORS

© 2017 CTU. All rights reserved.

Figure C.10: Login screen.

 Knowledge Base

Sign in

E-mail address 

We need your e-mail to reset the password.

Password

[Forgot password?](#)

---

ABOUT CTU | MEET THE CONTRIBUTORS

© 2017 CTU. All rights reserved.

Figure C.11: Forgotten password.



Knowledge Base

We have sent you an e-mail with  
instructions on how to reset your password.

Contact administrator

Figure C.12: Forgotten password - success message.



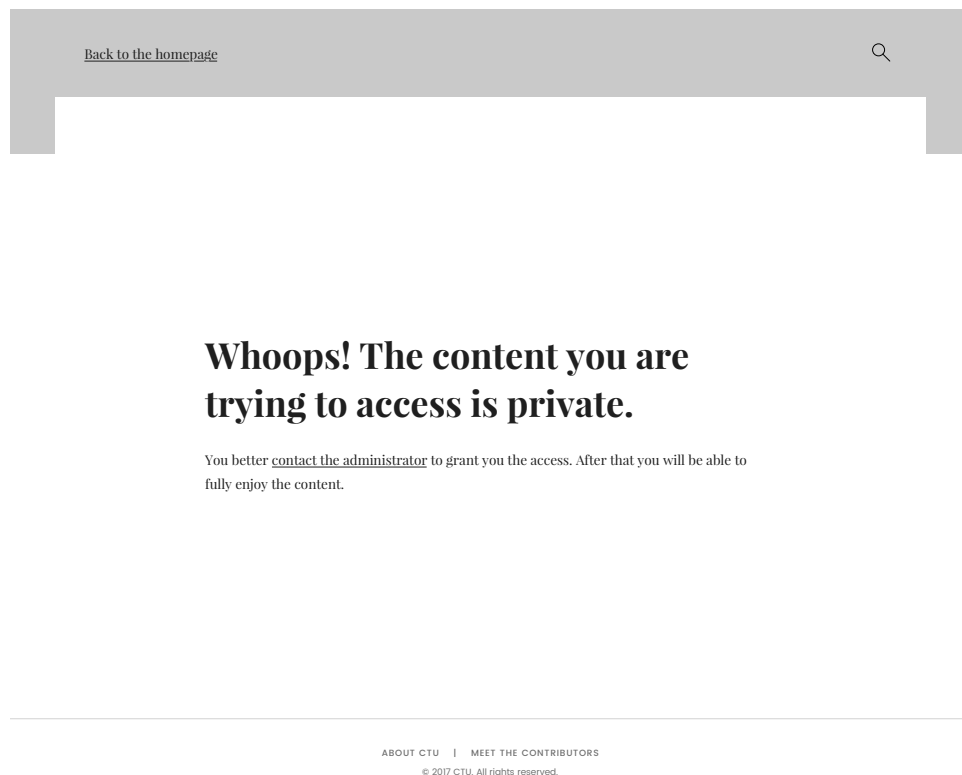


Figure C.13: Accessing a link that user does not have permission to access.

## C.4 Administration Interface

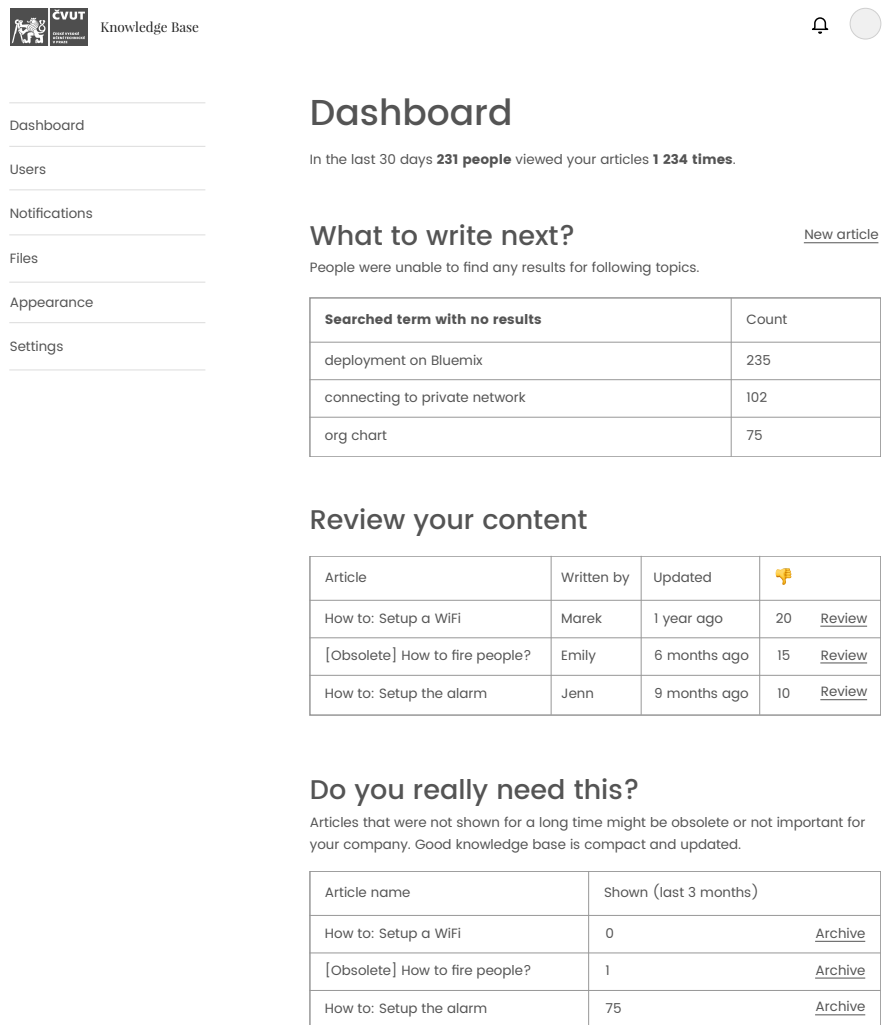





Figure C.14: Example of a wiki dashboard.

 Knowledge Base



Dashboard

Users

Notifications

Files

Appearance

Settings

## General

Name of the knowledge base

Usually the name of your company.

Claim your site

Must be lowercase characters.  
No spaces. Can't be changed  
after.

Delete Wiki

Figure C.15: General settings.

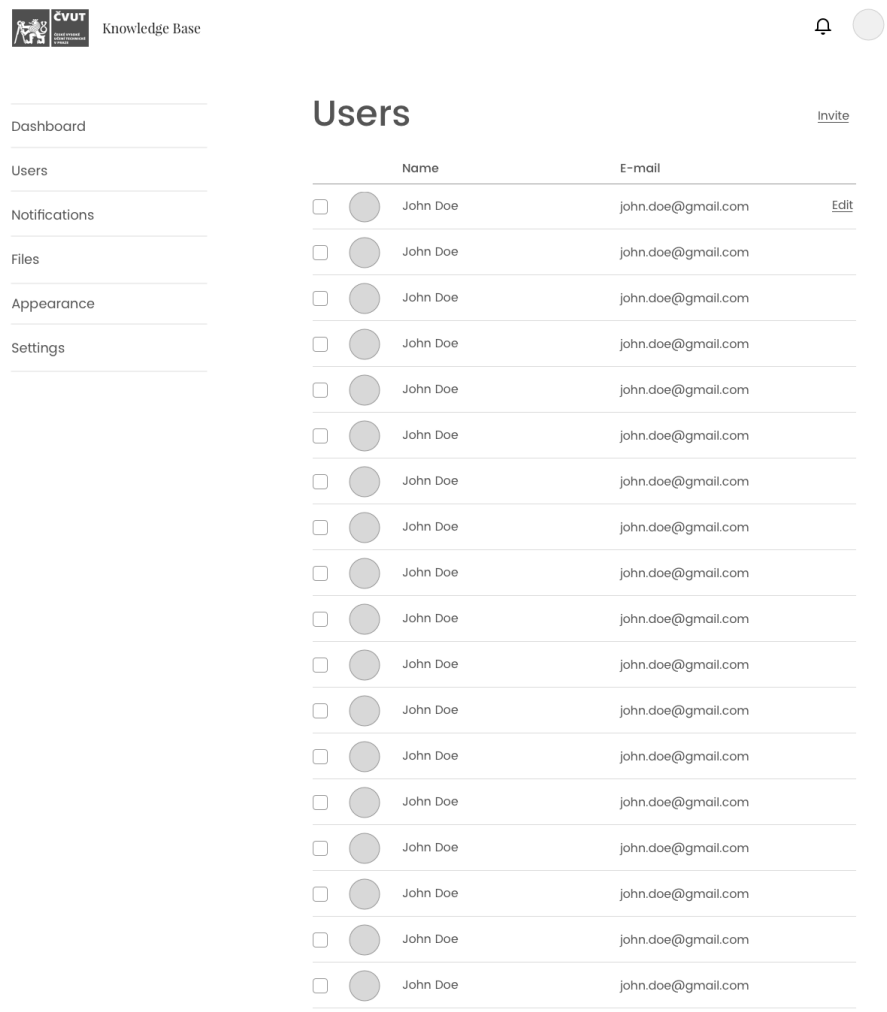





Figure C.16: List of wiki members.

 Knowledge Base



Dashboard

Users

Notifications

Files

Appearance

Settings

## Edit profile

Save

First Name

Marek

Last Name

Marek

Work e-mail address

Marek

Position

Product Designer

Location

San Francisco Office

Bio

Designer 21yo • Living in Prague 🇨🇪 • Originally from 🇬🇧 🇩🇪 Amateur self-taught photographer 🌍 Traveling the world 🧑 See the world' beautiful destinations

Role

Administrator ▼


Change Password



Change Password

Delete Account

Delete Account

Figure C.17: Editing profile.

 Knowledge Base



Dashboard

Users

Notifications

Files

Appearance

Settings

# Appearance

Brand Color

#215039

Logo

[Remove](#)  
[Upload new](#)

Favicon

[Remove](#)  
[Upload new](#)

Add links to your footer

Title

E.g. Our website	<a href="http://mycompany.co">http://mycompany.co</a>
E.g. Our website	<a href="http://mycompany.co">http://mycompany.co</a>
E.g. Our website	<a href="http://mycompany.co">http://mycompany.co</a>

Figure C.18: Adjusting the appearance of a wiki.

## D Source Code

**knowlywiki.zip** - The zip file with source code of the application can be found on the CD attached to this thesis. In order to install the application Ruby, the Rails Framework, a Web Server and a Database System are required.





## Bibliography

1. EBERSBACH, Anja; GLASER, Markus; HEIGL, Richard. *Wiki web collaboration*. Springer, 2006.
2. WEST, James A.; WEST, Margaret L. *Using wikis for online collaboration the power of the read-write Web*. Jossey-Bass, 2009.
3. CUNNINGHAM, Ward. *Wiki History* [online]. 2014 [visited on 2019-04-17]. Available from: <http://wiki.c2.com/?WikiHistory>.
4. LOWDERMILK, Travis. *User-Centered Design: A Developers Guide to Building User-Friendly Applications*. 2013.
5. HOOVER, Cole. *Human-Centered Design vs. Design-Thinking: How They're Different and How to Use Them Together to Create Lasting Change* [online]. 2018 [visited on 2019-04-17]. Available from: <https://blog.movingworlds.org/human-centered-design-vs-design-thinking-how-theyre-different-and-how-to-use-them-together-to-create-lasting-change/>.
6. BROWN, Tim. *Design Thinking* [online] [visited on 2019-04-17]. Available from: <http://hbr.org/2008/06/design-thinking/>.
7. GIBBONS, Sarah. *Design Thinking 101* [online]. 2016 [visited on 2019-04-23]. Available from: <https://www.nngroup.com/articles/design-thinking/>.
8. GOODMAN, Elizabeth; KUNIAVSKY, Mike; MOED, Andrea. *Observing the user experience: a practitioners guide to user research*. 2nd ed. Elsevier, 2012.
9. HARSHADEWA. *Here's why you should stop using Personas* [online] [visited on 2019-04-17]. Available from: <https://uxdesign.cc/heres-why-you-should-stop-using-personas-63c09a844e67>.
10. TRAYNOR, Des; ADAMS, Paul. *Intercom on Jobs-to-be-Done*. 1st ed. Intercom. Available also from: <https://www.intercom.com/books/jobs-to-be-done>.
11. DSDM. *MoSCoW Prioritisation* [online] [visited on 2019-04-17]. Available from: <https://www.agilebusiness.org/content/moscow-prioritisation>.

## BIBLIOGRAPHY

---

12. CLARKE, Ben. Why These Tech Companies Keep Running Thousands Of Failed Experiments [online] [visited on 2019-04-17]. Available from: <https://www.fastcompany.com/3063846/why-these-tech-companies-keep-running-thousands-of-failed>.
13. GOEL, Dhananjay. 8 Reasons to Incubate your next Startup (App) in Ruby on Rails [online] [visited on 2019-04-17]. Available from: <https://www.alphalogicinc.com/blog/8-reasons-to-incubate-your-next-startup-app-in-ruby-on-rails/>.
14. REJMAN, Michał. 40 Best Ruby On Rails Companies Websites [State For 2019] [online] [visited on 2019-04-17]. Available from: <https://ideamotive.co/blog/40-best-ruby-on-rails-companies-websites/>.
15. HARTL, Michael. *Ruby on Rails tutorial: learn web development with Rails*. Addison-Wesley, 2017.
16. SICHANUGRIST, Prem. *Getting Started with Rails* [online]. 2019 [visited on 2019-04-17]. Available from: [https://guides.rubyonrails.org/getting\\_started.html](https://guides.rubyonrails.org/getting_started.html).
17. FERNANDEZ, Obie; BOWKETT, Giles. *The Rails 5 way*. 4th ed. Addison-Wesley, 2018.
18. RODI, Alessandro. A Modern Web Application With Rails [online] [visited on 2019-04-17]. Available from: <https://medium.com/rubyinside/a-modern-web-application-with-rails-da3deb48014c>.
19. SAM RUBY David B. Copeland, Dave Thomas. *Agile Web Development with Rails 5.1*. 1st ed. Pragmatic Bookshelf, 2017.
20. OSTEZER; DRAKE, Mark. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems [online] [visited on 2019-04-17]. Available from: <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>.
21. KANE, Andrew. *Searchkick* [online]. 2018 [visited on 2019-04-17]. Available from: <https://github.com/ankane/searchkick>.
22. THOMAS, Jeremy; WALKER, Sunny. *Bulma: an alternative to Bootstrap* [online]. 2018 [visited on 2019-04-17]. Available from: <https://bulma.io/alternative-to-bootstrap/>.

## BIBLIOGRAPHY

---

23. GOODRICH, Glenn. Understanding the Model-View-Controller (MVC) Architecture in Rails [online] [visited on 2019-04-17]. Available from: <https://www.sitepoint.com/model-view-controller-mvc-architecture-rails/>.
24. PETROV, Igor. Building a multi-tenant app is easy...if you have an apartment! [online] [visited on 2019-04-17]. Available from: <https://medium.freecodecamp.org/building-a-multi-tenant-app-is-easy-if-you-have-an-apartment-3465f6eda85b>.
25. COSTA, Andre. What are Emoji's? How and When to Use Them [online] [visited on 2019-04-17]. Available from: <https://www.groovypost.com/howto/what-are-emojis-how-and-when-to-use-them/>.
26. LARDINOIS, Frederic. More than 3M businesses now pay for Google's G Suite [online] [visited on 2019-04-17]. Available from: <https://techcrunch.com/2017/01/26/more-than-3m-businesses-now-pay-for-googles-g-suite/>.
27. CLARK, Bryan. Google just passed 3M businesses paying for G Suite [online] [visited on 2019-04-17]. Available from: <https://thenextweb.com/insider/2017/01/27/google-just-passed-3m-businesses-paying-for-g-suite/>.
28. HALMAGEAN, Cezar. Feature Tests vs. Integration Tests vs. Unit Tests [online] [visited on 2019-05-22]. Available from: <https://mixandgo.com/learn/feature-tests-vs-integration-tests-vs-unit-tests-in-ruby-and-rails>.